User documentation
If you’re new to HPX you can get started with the Quick start guide. Don’t forget to read the Terminology section to learn about the most important concepts in HPX. The Examples give you a feel for how it is to write real HPX applications and the Manual contains detailed information about everything from building HPX to debugging it. There are links to blog posts and videos about HPX in Additional material.

If you can’t find what you’re looking for in the documentation, please:

- open an issue on GitHub\(^1\);
- contact us on IRC, the HPX channel on the C++ Slack\(^2\), or on our mailing list\(^3\); or
- read or ask questions tagged with HPX on StackOverflow\(^4\).

\(^1\) [https://github.com/STEllAR-GROUP/hpx/issues](https://github.com/STEllAR-GROUP/hpx/issues)
\(^2\) [https://cpplang.slack.com](https://cpplang.slack.com)
\(^3\) hpx-users@stellar.cct.lsu.edu
\(^4\) [https://stackoverflow.com/questions/tagged/hpx](https://stackoverflow.com/questions/tagged/hpx)
 CHAPTER 1

What is HPX?

HPX is a C++ Standard Library for Concurrency and Parallelism. It implements all of the corresponding facilities as defined by the C++ Standard. Additionally, in HPX we implement functionalities proposed as part of the ongoing C++ standardization process. We also extend the C++ Standard APIs to the distributed case. HPX is developed by the STEllAR group (see People).

The goal of HPX is to create a high quality, freely available, open source implementation of a new programming model for conventional systems, such as classic Linux based Beowulf clusters or multi-socket highly parallel SMP nodes. At the same time, we want to have a very modular and well designed runtime system architecture which would allow us to port our implementation onto new computer system architectures. We want to use real-world applications to drive the development of the runtime system, coining out required functionalities and converging onto a stable API which will provide a smooth migration path for developers.

The API exposed by HPX is not only modeled after the interfaces defined by the C++11/14/17/20 ISO standard. It also adheres to the programming guidelines used by the Boost collection of C++ libraries. We aim to improve the scalability of today’s applications and to expose new levels of parallelism which are necessary to take advantage of the exascale systems of the future.
What’s so special about *HPX*?

- HPX exposes a uniform, standards-oriented API for ease of programming parallel and distributed applications.
- It enables programmers to write fully asynchronous code using hundreds of millions of threads.
- HPX provides unified syntax and semantics for local and remote operations.
- HPX makes concurrency manageable with dataflow and future based synchronization.
- It implements a rich set of runtime services supporting a broad range of use cases.
- HPX exposes a uniform, flexible, and extendable performance counter framework which can enable runtime adaptivity.
- It is designed to solve problems conventionally considered to be scaling-impaired.
- HPX has been designed and developed for systems of any scale, from hand-held devices to very large scale systems.
- It is the first fully functional implementation of the ParalleX execution model.
- HPX is published under a liberal open-source license and has an open, active, and thriving developer community.

2.1 Why *HPX*?

Current advances in high performance computing (HPC) continue to suffer from the issues plaguing parallel computation. These issues include, but are not limited to, ease of programming, inability to handle dynamically changing workloads, scalability, and efficient utilization of system resources. Emerging technological trends such as multi-core processors further highlight limitations of existing parallel computation models. To mitigate the aforementioned problems, it is necessary to rethink the approach to parallelization models. ParalleX contains mechanisms such as multi-threading, *parcels*, *global name space* support, percolation and *local control objects* (*LCO*). By design, ParalleX overcomes limitations of current models of parallelism by alleviating contention, latency, overhead and starvation. With ParalleX, it is further possible to increase performance by at least an order of magnitude on challenging parallel algorithms, e.g., dynamic directed graph algorithms and adaptive mesh refinement methods for astrophysics. An additional benefit of ParalleX is fine-grained control of power usage, enabling reductions in power consumption.
2.1.1 ParalleX—a new execution model for future architectures

ParalleX is a new parallel execution model that offers an alternative to the conventional computation models, such as message passing. ParalleX distinguishes itself by:

- Split-phase transaction model
- Message-driven
- Distributed shared memory (not cache coherent)
- Multi-threaded
- Futures synchronization
- Local Control Objects (LCOs)
- Synchronization for anonymous producer-consumer scenarios
- Percolation (pre-staging of task data)

The ParalleX model is intrinsically latency hiding, delivering an abundance of variable-grained parallelism within a hierarchical namespace environment. The goal of this innovative strategy is to enable future systems delivering very high efficiency, increased scalability and ease of programming. ParalleX can contribute to significant improvements in the design of all levels of computing systems and their usage from application algorithms and their programming languages to system architecture and hardware design together with their supporting compilers and operating system software.

2.1.2 What is HPX?

High Performance ParalleX (HPX) is the first runtime system implementation of the ParalleX execution model. The HPX runtime software package is a modular, feature-complete, and performance-oriented representation of the ParalleX execution model targeted at conventional parallel computing architectures, such as SMP nodes and commodity clusters. It is academically developed and freely available under an open source license. We provide HPX to the community for experimentation and application to achieve high efficiency and scalability for dynamic adaptive and irregular computational problems. HPX is a C++ library that supports a set of critical mechanisms for dynamic adaptive resource management and lightweight task scheduling within the context of a global address space. It is solidly based on many years of experience in writing highly parallel applications for HPC systems.

The two-decade success of the communicating sequential processes (CSP) execution model and its message passing interface (MPI) programming model have been seriously eroded by challenges of power, processor core complexity, multi-core sockets, and heterogeneous structures of GPUs. Both efficiency and scalability for some current (strong scaled) applications and future Exascale applications demand new techniques to expose new sources of algorithm parallelism and exploit unused resources through adaptive use of runtime information.

The ParalleX execution model replaces CSP to provide a new computing paradigm embodying the governing principles for organizing and conducting highly efficient scalable computations greatly exceeding the capabilities of today’s problems. HPX is the first practical, reliable, and performance-oriented runtime system incorporating the principal concepts of the ParalleX model publicly provided in open source release form.

HPX is designed by the STE||AR\(^5\) Group (Systems Technology, Emergent Parallelism, and Algorithm Research) at Louisiana State University (LSU)\(^6\)’s Center for Computation and Technology (CCT)\(^7\) to enable developers to exploit the full processing power of many-core systems with an unprecedented degree of parallelism. STE||AR\(^8\) is a research group focusing on system software solutions and scientific application development for hybrid and many-core hardware architectures.

\(^5\) https://stellar-group.org
\(^6\) https://www.lsu.edu
\(^7\) https://www.cct.lsu.edu
\(^8\) https://stellar-group.org
For more information about the STEllAR\textsuperscript{9} Group, see \textit{People}.

### 2.1.3 What makes our systems slow?

Estimates say that we currently run our computers at well below 100% efficiency. The theoretical peak performance (usually measured in FLOPS\textsuperscript{10}—floating point operations per second) is much higher than any practical peak performance reached by any application. This is particularly true for highly parallel hardware. The more hardware parallelism we provide to an application, the better the application must scale in order to efficiently use all the resources of the machine. Roughly speaking, we distinguish two forms of scalability: strong scaling (see \textit{Amdahl’s Law}\textsuperscript{11}) and weak scaling (see \textit{Gustafson’s Law}\textsuperscript{12}). Strong scaling is defined as how the solution time varies with the number of processors for a fixed total problem size. It gives an estimate of how much faster we can solve a particular problem by throwing more resources at it. Weak scaling is defined as how the solution time varies with the number of processors for a fixed problem size per processor. In other words, it defines how much more data can we process by using more hardware resources.

In order to utilize as much hardware parallelism as possible an application must exhibit excellent strong and weak scaling characteristics, which requires a high percentage of work executed in parallel, i.e., using multiple threads of execution. Optimally, if you execute an application on a hardware resource with \(N\) processors it either runs \(N\) times faster or it can handle \(N\) times more data. Both cases imply 100% of the work is executed on all available processors in parallel. However, this is just a theoretical limit. Unfortunately, there are more things that limit scalability, mostly inherent to the hardware architectures and the programming models we use. We break these limitations into four fundamental factors that make our systems \textit{SLOW}:

- Starvation occurs when there is insufficient concurrent work available to maintain high utilization of all resources.
- Latencies are imposed by the time-distance delay intrinsic to accessing remote resources and services.
- Overhead is work required for the management of parallel actions and resources on the critical execution path, which is not necessary in a sequential variant.
- Waiting for contention resolution is the delay due to the lack of availability of oversubscribed shared resources.

Each of those four factors manifests itself in multiple and different ways; each of the hardware architectures and programming models expose specific forms. However, the interesting part is that all of them are limiting the scalability of applications no matter what part of the hardware jungle we look at. Hand-holds, PCs, supercomputers, or the cloud, all suffer from the reign of the 4 horsemen: Starvation, Latency, Overhead, and Contention. This realization is very important as it allows us to derive the criteria for solutions to the scalability problem from first principles, and it allows us to focus our analysis on very concrete patterns and measurable metrics. Moreover, any derived results will be applicable to a wide variety of targets.

### 2.1.4 Technology demands new response

Today’s computer systems are designed based on the initial ideas of \textit{John von Neumann}\textsuperscript{13}, as published back in 1945, and later extended by the \textit{Harvard architecture}\textsuperscript{14}. These ideas form the foundation, the execution model, of computer systems we use currently. However, a new response is required in the light of the demands created by today’s technology.

So, what are the overarching objectives for designing systems allowing for applications to scale as they should? In our opinion, the main objectives are:

- Performance: as previously mentioned, scalability and efficiency are the main criteria people are interested in.

\textsuperscript{9} https://stellar-group.org
\textsuperscript{10} http://en.wikipedia.org/wiki/FLOPS
\textsuperscript{11} http://en.wikipedia.org/wiki/Amdahl%27s_law
\textsuperscript{12} http://en.wikipedia.org/wiki/Gustafson%27s_law
\textsuperscript{13} http://qss.stanford.edu/~godfrey/vonNeumann/vnedvac.pdf
\textsuperscript{14} http://en.wikipedia.org/wiki/Harvard_architecture
• Fault tolerance: the low expected mean time between failures (MTBF\(^{15}\)) of future systems requires embracing faults, not trying to avoid them.
• Power: minimizing energy consumption is a must as it is one of the major cost factors today, and will continue to rise in the future.
• Generality: any system should be usable for a broad set of use cases.
• Programmability: for programmer this is a very important objective, ensuring long term platform stability and portability.

What needs to be done to meet those objectives, to make applications scale better on tomorrow’s architectures? Well, the answer is almost obvious: we need to devise a new execution model—a set of governing principles for the holistic design of future systems—targeted at minimizing the effect of the outlined SLOW factors. Everything we create for future systems, every design decision we make, every criteria we apply, have to be validated against this single, uniform metric. This includes changes in the hardware architecture we prevalently use today, and it certainly involves new ways of writing software, starting from the operating system, runtime system, compilers, and at the application level. However, the key point is that all those layers have to be co-designed; they are interdependent and cannot be seen as separate facets. The systems we have today have been evolving for over 50 years now. All layers function in a certain way, relying on the other layers to do so. But we do not have the time to wait another 50 years for a new coherent system to evolve. The new paradigms are needed now—therefore, co-design is the key.

2.1.5 Governing principles applied while developing HPX

As it turn out, we do not have to start from scratch. Not everything has to be invented and designed anew. Many of the ideas needed to combat the 4 horsemen already exist, many for more than 30 years. All it takes is to gather them into a coherent approach. We’ll highlight some of the derived principles we think to be crucial for defeating SLOW. Some of those are focused on high-performance computing, others are more general.

2.1.6 Focus on latency hiding instead of latency avoidance

It is impossible to design a system exposing zero latencies. In an effort to come as close as possible to this goal many optimizations are mainly targeted towards minimizing latencies. Examples for this can be seen everywhere, such as low latency network technologies like InfiniBand\(^{16}\), caching memory hierarchies in all modern processors, the constant optimization of existing MPI\(^{17}\) implementations to reduce related latencies, or the data transfer latencies intrinsic to the way we use GPGPUs\(^{18}\) today. It is important to note that existing latencies are often tightly related to some resource having to wait for the operation to be completed. At the same time it would be perfectly fine to do some other, unrelated work in the meantime, allowing the system to hide the latencies by filling the idle-time with useful work. Modern systems already employ similar techniques (pipelined instruction execution in the processor cores, asynchronous input/output operations, and many more). What we propose is to go beyond anything we know today and to make latency hiding an intrinsic concept of the operation of the whole system stack.

2.1.7 Embrace fine-grained parallelism instead of heavyweight threads

If we plan to hide latencies even for very short operations, such as fetching the contents of a memory cell from main memory (if it is not already cached), we need to have very lightweight threads with extremely short context switching times, optimally executable within one cycle. Granted, for mainstream architectures, this is not possible today (even if we already have special machines supporting this mode of operation, such as the Cray XMT\(^{19}\)). For conventional systems, however, the smaller the overhead of a context switch and the finer the granularity of the threading system, the better will be the overall system utilization and its efficiency. For today’s architectures we already see a flurry of libraries providing exactly this type of functionality: non-pre-emptive, task-queue based parallelization solutions, such

\(^{15}\) http://en.wikipedia.org/wiki/Mean_time_between_failures
\(^{16}\) http://en.wikipedia.org/wiki/InfiniBand
\(^{17}\) https://en.wikipedia.org/wiki/Message_Passing_Interface
\(^{18}\) http://en.wikipedia.org/wiki/GPGPU
\(^{19}\) http://en.wikipedia.org/wiki/Cray_XMT
The possibility to suspend a current task if some preconditions for its execution are not met (such as waiting for I/O or the result of a different task), seamlessly switching to any other task which can continue, and to reschedule the initial task after the required result has been calculated, which makes the implementation of latency hiding almost trivial.

2.1.8 Rediscover constraint-based synchronization to replace global barriers

The code we write today is riddled with implicit (and explicit) global barriers. By “global barriers,” we mean the synchronization of the control flow between several (very often all) threads (when using OpenMP) or processes (MPI). For instance, an implicit global barrier is inserted after each loop parallelized using OpenMP as the system synchronizes the threads used to execute the different iterations in parallel. In MPI each of the communication steps imposes an explicit barrier onto the execution flow as (often all) nodes have to be synchronized. Each of those barriers is like the eye of a needle the overall execution is forced to be squeezed through. Even minimal fluctuations in the execution times of the parallel threads (jobs) causes them to wait. Additionally, it is often only one of the executing threads that performs the actual reduce operation, which further impedes parallelism. A closer analysis of a couple of key algorithms used in science applications reveals that these global barriers are not always necessary. In many cases it is sufficient to synchronize a small subset of the threads. Any operation should proceed whenever the preconditions for its execution are met, and only those. Usually there is no need to wait for iterations of a loop to finish before you can continue calculating other things; all you need is to complete the iterations that produce the required results for the next operation. Good bye global barriers, hello constraint based synchronization! People have been trying to build this type of computing (and even computers) since the 1970s. The theory behind what they did is based on ideas around static and dynamic dataflow. There are certain attempts today to get back to those ideas and to incorporate them with modern architectures. For instance, a lot of work is being done in the area of constructing dataflow-oriented execution trees. Our results show that employing dataflow techniques in combination with the other ideas, as outlined herein, considerably improves scalability for many problems.

2.1.9 Adaptive locality control instead of static data distribution

While this principle seems to be a given for single desktop or laptop computers (the operating system is your friend), it is everything but ubiquitous on modern supercomputers, which are usually built from a large number of separate nodes (i.e., Beowulf clusters), tightly interconnected by a high-bandwidth, low-latency network. Today’s prevalent programming model for those is MPI, which does not directly help with proper data distribution, leaving it to the programmer to decompose the data to all of the nodes the application is running on. There are a couple of specialized languages and programming environments based on PGAS (Partitioned Global Address Space) designed to overcome this limitation, such as Chapel, X10, UPC, or Fortress. However, all systems based on PGAS rely on static data distribution. This works fine as long as this static data distribution does not result in heterogeneous workload distributions or other resource utilization imbalances. In a distributed system these imbalances can be mitigated by migrating part of the application data to different localities (nodes). The only framework supporting (limited) migration today is Charm++. The first attempts towards solving related problem go back decades as well, a good example is the Linda coordination language. Nevertheless, none of the other mentioned systems support data migration today, which forces the users to either rely on static data distribution and live with the related performance hits or to implement everything themselves, which is very tedious and difficult. We believe that the only viable way to flexibly...
support dynamic and adaptive *locality* control is to provide a global, uniform address space to the applications, even on distributed systems.

### 2.1.10 Prefer moving work to the data over moving data to the work

For the best performance it seems obvious to minimize the amount of bytes transferred from one part of the system to another. This is true on all levels. At the lowest level we try to take advantage of processor memory caches, thus, minimizing memory latencies. Similarly, we try to amortize the data transfer time to and from GPGPUs\(^{34}\) as much as possible. At high levels we try to minimize data transfer between different nodes of a cluster or between different virtual machines on the cloud. Our experience (well, it’s almost common wisdom) shows that the amount of bytes necessary to encode a certain operation is very often much smaller than the amount of bytes encoding the data the operation is performed upon. Nevertheless, we still often transfer the data to a particular place where we execute the operation just to bring the data back to where it came from afterwards. As an example let’s look at the way we usually write our applications for clusters using MPI. This programming model is all about data transfer between nodes. MPI is the prevalent programming model for clusters, and it is fairly straightforward to understand and to use. Therefore, we often write applications in a way that accommodates this model, centered around data transfer. These applications usually work well for smaller problem sizes and for regular data structures. The larger the amount of data we have to churn and the more irregular the problem domain becomes, the worse the overall machine utilization and the (strong) scaling characteristics become. While it is not impossible to implement more dynamic, data driven, and asynchronous applications using MPI, it is somewhat difficult to do so. At the same time, if we look at applications that prefer to execute the code close to the *locality* where the data was placed, i.e., utilizing active messages (for instance based on Charm++\(^{35}\)), we see better asynchrony, simpler application codes, and improved scaling.

### 2.1.11 Favor message driven computation over message passing

Today’s prevalently used programming model on parallel (multi-node) systems is MPI. It is based on message passing, as the name implies, which means that the receiver has to be aware of a message about to come in. Both codes, the sender and the receiver, have to synchronize in order to perform the communication step. Even the newer, asynchronous interfaces require explicitly coding the algorithms around the required communication scheme. As a result, everything but the most trivial MPI applications spends a considerable amount of time waiting for incoming messages, thus, causing starvation and latencies to impede full resource utilization. The more complex and more dynamic the data structures and algorithms become, the larger the adverse effects. The community discovered message-driven and data-driven methods of implementing algorithms a long time ago, and systems such as Charm++\(^{36}\) have already integrated active messages demonstrating the validity of the concept. Message-driven computation allows for sending messages without requiring the receiver to actively wait for them. Any incoming message is handled asynchronously and triggers the encoded action by passing along arguments and—possibly—continuations. HPX combines this scheme with work-queue based scheduling as described above, which allows the system to almost completely overlap any communication with useful work, thereby minimizing latencies.

### 2.2 Quick start

This section is intended to get you to the point of running a basic *HPX* program as quickly as possible. To that end we skip many details but instead give you hints and links to more details along the way.

We assume that you are on a Unix system with access to reasonably recent packages. You should have `cmake` and `make` available for the build system (`pkg-config` is also supported, see Using *HPX* with `pkg-config`).

---


\(^{35}\) [https://charm.cs.uiuc.edu/](https://charm.cs.uiuc.edu/)

\(^{36}\) [https://charm.cs.uiuc.edu/](https://charm.cs.uiuc.edu/)
2.2.1 Getting HPX

Download a tarball of the latest release from HPX Downloads\textsuperscript{37} and unpack it or clone the repository directly using git:

\begin{verbatim}
git clone https://github.com/STEllAR-GROUP/hpx.git
\end{verbatim}

It is also recommended that you check out the latest stable tag:

\begin{verbatim}
git checkout 1.4.1
\end{verbatim}

2.2.2 HPX dependencies

The minimum dependencies needed to use HPX are Boost\textsuperscript{38} and Portable Hardware Locality (HWLOC)\textsuperscript{39}. If these are not available through your system package manager, see Installing Boost and Installing Hwloc for instructions on how to build them yourself. In addition to Boost\textsuperscript{30} and Portable Hardware Locality (HWLOC)\textsuperscript{41}, it is recommended that you don’t use the system allocator, but instead use either tcmalloc from google-perftools\textsuperscript{42} (default) or jemalloc\textsuperscript{43} for better performance. If you would like to try HPX without a custom allocator at this point you can configure HPX to use the system allocator in the next step.

A full list of required and optional dependencies, including recommended versions is available at Prerequisites.

2.2.3 Building HPX

Once you have the source code and the dependencies, set up a separate build directory and configure the project. Assuming all your dependencies are in paths known to CMake, the following gets you started:

\begin{verbatim}
# In the HPX source directory
mkdir build && cd build
cmake -DCMAKE_INSTALL_PREFIX=/install/path ..
make install
\end{verbatim}

This will build the core HPX libraries and examples, and install them to your chosen location. If you want to install HPX to system folders simply leave out the CMAKE_INSTALL_PREFIX option. This may take a while. To speed up the process launch more jobs by passing the \texttt{-jN} option to make.

\textbf{Tip:} Do not set only \texttt{-j} (i.e. \texttt{-j} without an explicit number of jobs) unless you have a lot of memory available on your machine.

\textbf{Tip:} If you want to change CMake\textsuperscript{44} variables for your build it is usually a good idea to start with a clean build directory to avoid configuration problems. It is especially important that you use a clean build directory when changing between Release and Debug modes.

If your dependencies are in custom locations you may need to tell CMake\textsuperscript{45} where to find them by passing one or more of the following options to CMake:\textsuperscript{46}:

\textsuperscript{37} https://stellar-group.org/downloads/
\textsuperscript{38} https://www.boost.org/
\textsuperscript{39} https://www.open-mpi.org/projects/hwloc/
\textsuperscript{40} https://www.boost.org/
\textsuperscript{41} https://www.open-mpi.org/projects/hwloc/
\textsuperscript{42} https://code.google.com/p/gperftools
\textsuperscript{43} http://jemalloc.net
\textsuperscript{44} https://www.cmake.org
\textsuperscript{45} https://www.cmake.org
\textsuperscript{46} https://www.cmake.org
If you want to try HPX without using a custom allocator pass `-DHPX_WITH_MALLOC=system` to CMake\(^47\).

**Important:** If you are building HPX for a system with more than 64 processing units you must change the CMake variables `HPX_WITH_MORE_THAN_64_THREADS` (to `On`) and `HPX_WITH_MAX_CPU_COUNT` (to a value at least as big as the number of (virtual) cores on your system).

To build the tests run `make tests`. To run the tests run either `make test` or use `ctest` for more control over which tests to run. You can run single tests for example with `ctest --output-on-failure -R tests.unit.parallel.algorithms.for_loop` or a whole group of tests with `ctest --output-on-failure -R tests.unit`.

If you did not run `make install` earlier do so now or build the `hello_world_1` example by running:

```
make hello_world_1
```

HPX executables end up in the `bin` directory in your build directory. You can now run `hello_world_1` and should see the following output:

```
./bin/hello_world_1
Hello World!
```

You’ve just run an example which prints Hello World! from the HPX runtime. The source for the example is in `examples/quickstart/hello_world_1.cpp`. The `hello_world_distributed` example (also available in the `examples/quickstart` directory) is a distributed hello world program which is described in Remote execution with actions: Hello world. It provides a gentle introduction to the distributed aspects of HPX.

**Tip:** Most build targets in HPX have two names: a simple name and a hierarchical name corresponding to what type of example or test the target is. If you are developing HPX it is often helpful to run `make help` to get a list of available targets. For example, `make help | grep hello_world` outputs the following:

```
... examples.quickstart.hello_world_2
... hello_world_2
... examples.quickstart.hello_world_1
... hello_world_1
... examples.quickstart.hello_world_distributed
... hello_world_distributed
```

It is also possible to build e.g. all quickstart examples using `make examples.quickstart`.

### 2.2.4 Hello, World!

The following `CMakeLists.txt` is a minimal example of what you need in order to build an executable using CMake\(^48\) and HPX:

\(^47\) [https://www.cmake.org](https://www.cmake.org)
\(^48\) [https://www.cmake.org](https://www.cmake.org)
```cmake
cmake_minimum_required(VERSION 3.3.2)
project(my_hpx_project CXX)
find_package(HPX REQUIRED)
add_hpx_executable(my_hpx_program
    SOURCES main.cpp
    COMPONENT_DEPENDENCIES iostreams)
```

**Note:** You will most likely have more than one main.cpp file in your project. See the section on *Using HPX with CMake-based projects* for more details on how to use add_hpx_executable.

**Note:** COMPONENT_DEPENDENCIES iostreams is optional for a minimal project but lets us use the HPX equivalent of std::cout, i.e. the HPX The HPX I/O-streams component functionality in our application.

Create a new project directory and a CMakeLists.txt with the contents above. Also create a main.cpp with the contents below.

```cpp
#include <hpx/hpx_main.hpp>
#include <hpx/include/iostreams.hpp>

int main()
{
    // Say hello to the world!
    hpx::cout << "Hello World!\n" << hpx::flush;
    return 0;
}
```

Then, in your project directory run the following:

```bash
mkdir build && cd build
cmake -DCMAKE_PREFIX_PATH=/path/to/hpx/installation ..
make all
./my_hpx_program
```

The program looks almost like a regular C++ hello world with the exception of the two includes and hpx::cout. When you include hpx_main.hpp some things will be done behind the scenes to make sure that main actually gets launched on the HPX runtime. So while it looks almost the same you can now use futures, async, parallel algorithms and more which make use of the HPX runtime with lightweight threads. hpx::cout is a replacement for std::cout to make sure printing never blocks a lightweight thread. You can read more about hpx::cout in *The HPX I/O-streams component*. If you rebuild and run your program now you should see the familiar Hello World!:

```
./my_hpx_program
Hello World!
```

**Note:** You do not have to let HPX take over your main function like in the example. You can instead keep your normal main function, and define a separate hpx_main function which acts as the entry point to the HPX runtime. In that case you start the HPX runtime explicitly by calling hpx::init:

```cpp
// Copyright (c) 2007-2012 Hartmut Kaiser
//
// SPDX-License-Identifier: BSL-1.0
```

(continues on next page)
You can also use `hpx::start` and `hpx::stop` for a non-blocking alternative, or use `hpx::resume` and `hpx::suspend` if you need to combine `HPX` with other runtimes.

See *Starting the HPX runtime* for more details on how to initialize and run the `HPX` runtime.

---

**Caution:** When including `hpx_main.hpp` the user-defined `main` gets renamed and the real `main` function is defined by `HPX`. This means that the user-defined `main` must include a return statement, unlike the real `main`. If you do not include the return statement you may end up with confusing compile time errors mentioning `user_main` or even runtime errors.

### 2.2.5 Writing task-based applications

So far we haven’t done anything that can’t be done using the C++ standard library. In this section we will give a short overview of what you can do with `HPX` on a single node. The essence is to avoid global synchronization and break up your application into small, composable tasks whose dependencies control the flow of your application. Remember, however, that `HPX` allows you to write distributed applications similarly to how you would write applications for a single node (see *Why HPX?* and *Writing distributed HPX applications*).

If you are already familiar with `async` and `futures` from the C++ standard library, the same functionality is available in `HPX`.

The following terminology is essential when talking about task-based C++ programs:

- **lightweight thread:** Essential for good performance with task-based programs. Lightweight refers to smaller stacks and faster context switching compared to OS-threads. Smaller overheads allow the program to be broken up into smaller tasks, which in turns helps the runtime fully utilize all processing units.

- **async:** The most basic way of launching tasks asynchronously. Returns a `future<T>`.

- **future<T>:** Represents a value of type `T` that will be ready in the future. The value can be retrieved with `get` (blocking) and one can check if the value is ready with `is_ready` (non-blocking).
• shared_future<T>: Same as future<T> but can be copied (similar to std::unique_ptr vs std::shared_ptr).

• continuation: A function that is to be run after a previous task has run (represented by a future). then is a method of future<T> that takes a function to run next. Used to build up dataflow DAGs (directed acyclic graphs). shared_futures help you split up nodes in the DAG and functions like when_all help you join nodes in the DAG.

The following example is a collection of the most commonly used functionality in HPX:

```cpp
#include <hpx/hpx_main.hpp>
#include <hpx/include/iostreams.hpp>
#include <hpx/include/lcos.hpp>
#include <hpx/include/parallel_generate.hpp>
#include <hpx/include/parallel_sort.hpp>

#include <random>
#include <vector>

void final_task(hpx::future<hpx::util::tuple<hpx::future<double>, hpx::future<void>>>)
{
    hpx::cout << "in final_task" << hpx::endl;
}

// Avoid ABI incompatibilities between C++11/C++17 as std::rand has exception
// specification in libstdc++.
int rand_wrapper()
{
    return std::rand();
}

int main(int, char**)
{
    // A function can be launched asynchronously. The program will not block
    // here until the result is available.
    hpx::future<int> f = hpx::async([]() { return 42; });
    hpx::cout << "Just launched a task!" << hpx::endl;

    // Use get to retrieve the value from the future. This will block this task
    // until the future is ready, but the HPX runtime will schedule other tasks
    // if there are tasks available.
    hpx::cout << "f contains " << f.get() << hpx::endl;

    // Let's launch another task.
    hpx::future<double> g = hpx::async([]() { return 3.14; });

    // Tasks can be chained using the then method. The continuation takes the
    // future as an argument.
    hpx::future<double> result = g.then([](hpx::future<double>&& gg)
    {
        // This function will be called once g is ready. gg is g moved
        // into the continuation.
        return gg.get() * 42.0 * 42.0;
    });

    // You can check if a future is ready with the is_ready method.
    hpx::cout << "Result is ready? " << result.is_ready() << hpx::endl;
}
```

(continues on next page)
// You can launch other work in the meantime. Let’s sort a vector.
std::vector<int> v(1000000);

// We fill the vector synchronously and sequentially.
hpx::parallel::generate(hpx::parallel::execution::seq,
    std::begin(v), std::end(v), &rand_wrapper);

// We can launch the sort in parallel and asynchronously.
hpx::future<void> done_sorting =
    hpx::parallel::sort(
        hpx::parallel::execution::par( // In parallel.
            hpx::parallel::execution::task), // Asynchronously.
        std::begin(v),
        std::end(v));

// We launch the final task when the vector has been sorted and result is
// ready using when_all.
auto all = hpx::when_all(result, done_sorting).then(&final_task);

// We can wait for all to be ready.
all.wait();

// all must be ready at this point because we waited for it to be ready.
hpx::cout <<
    (all.is_ready() ? "all is ready!" : "all is not ready...") << hpx::endl;

return hpx::finalize();
}

Try copying the contents to your main.cpp file and look at the output. It can be a good idea to go through the program step by step with a debugger. You can also try changing the types or adding new arguments to functions to make sure you can get the types to match. The type of the then method can be especially tricky to get right (the continuation needs to take the future as an argument).

Note: HPX programs accept command line arguments. The most important one is --hpx:threads=N to set the number of OS-threads used by HPX. HPX uses one thread per core by default. Play around with the example above and see what difference the number of threads makes on the sort function. See Launching and configuring HPX applications for more details on how and what options you can pass to HPX.

Tip: The example above used the construction hpx::when_all(...).then(...). For convenience and performance it is a good idea to replace uses of hpx::when_all(...).then(...) with dataflow. See Dataflow: Interest calculator for more details on dataflow.

Tip: If possible, prefer to use the provided parallel algorithms instead of writing your own implementation. This can save you time and the resulting program is often faster.

2.2.6 Next steps
If you haven’t done so already, reading the Terminology section will help you get familiar with the terms used in HPX. The Examples section contains small, self-contained walkthroughs of example HPX programs. The Local to remote:
1D stencil example is a thorough, realistic example starting from a single node implementation and going stepwise to a distributed implementation.

The Manual contains detailed information on writing, building and running HPX applications.

### 2.3 Terminology

This section gives definitions for some of the terms used throughout the HPX documentation and source code.

**Locality** A locality in HPX describes a synchronous domain of execution, or the domain of bounded upper response time. This normally is just a single node in a cluster or a NUMA domain in an SMP machine.

**Active Global Address Space (AGAS)** HPX incorporates a global address space. Any executing thread can access any object within the domain of the parallel application with the caveat that it must have appropriate access privileges. The model does not assume that global addresses are cache coherent; all loads and stores will deal directly with the site of the target object. All global addresses within a Synchronous Domain are assumed to be cache coherent for those processor cores that incorporate transparent caches. The Active Global Address Space used by HPX differs from research PGAS models. Partitioned Global Address Space is passive in their means of address translation. Copy semantics, distributed compound operations, and affinity relationships are some of the global functionality supported by AGAS.

**Process** The concept of the “process” in HPX is extended beyond that of either sequential execution or communicating sequential processes. While the notion of process suggests action (as do “function” or “subroutine”) it has a further responsibility of context, that is, the logical container of program state. It is this aspect of operation that process is employed in HPX. Furthermore, referring to “parallel processes” in HPX designates the presence of parallelism within the context of a given process, as well as the coarse grained parallelism achieved through concurrency of multiple processes of an executing user job. HPX processes provide a hierarchical name space within the framework of the active global address space and support multiple means of internal state access from external sources.

**Parcel** The Parcel is a component in HPX that communicates data, invokes an action at a distance, and distributes flow-control through the migration of continuations. Parcels bridge the gap of asynchrony between synchronous domains while maintaining symmetry of semantics between local and global execution. Parcels enable message-driven computation and may be seen as a form of “active messages”. Other important forms of message-driven computation predating active messages include dataflow tokens, the J-machine’s support for remote method instantiation, and at the coarse grained variations of Unix remote procedure calls, among others. This enables work to be moved to the data as well as performing the more common action of bringing data to the work. A parcel can cause actions to occur remotely and asynchronously, among which are the creation of threads at different system nodes or synchronous domains.

**Local Control Object (LCO)** A local control object (sometimes called a lightweight control object) is a general term for the synchronization mechanisms used in HPX. Any object implementing a certain concept can be seen as an LCO. This concept encapsulates the ability to be triggered by one or more events which when taking the object into a predefined state will cause a thread to be executed. This could either create a new thread or resume an existing thread.

The LCO is a family of synchronization functions potentially representing many classes of synchronization constructs, each with many possible variations and multiple instances. The LCO is sufficiently general that it can subsume the functionality of conventional synchronization primitives such as spinlocks, mutexes, semaphores, and global barriers. However due to the rich concept an LCO can represent powerful synchronization and control
functionality not widely employed, such as dataflow and futures (among others), which open up enormous opportunities for rich diversity of distributed control and operation.

See Using LCOs for more details on how to use LCOs in HPX.

**Action** An action is a function that can be invoked remotely. In HPX a plain function can be made into an action using a macro. See Applying actions for details on how to use actions in HPX.

**Component** A component is a C++ object which can be accessed remotely. A component can also contain member functions which can be invoked remotely. These are referred to as component actions. See Writing components for details on how to use components in HPX.

### 2.4 Examples

The following sections analyze some examples to help you get familiar with the HPX style of programming. We start off with simple examples that utilize basic HPX elements and then begin to expose the reader to the more complex and powerful HPX concepts.

#### 2.4.1 Asynchronous execution with hpx::async: Fibonacci

The Fibonacci sequence is a sequence of numbers starting with 0 and 1 where every subsequent number is the sum of the previous two numbers. In this example, we will use HPX to calculate the value of the n-th element of the Fibonacci sequence. In order to compute this problem in parallel, we will use a facility known as a future.

As shown in the Fig. ?? below, a future encapsulates a delayed computation. It acts as a proxy for a result initially not known, most of the time because the computation of the result has not completed yet. The future synchronizes the access of this value by optionally suspending any HPX-threads requesting the result until the value is available. When a future is created, it spawns a new HPX-thread (either remotely with a parcel or locally by placing it into the thread queue) which, when run, will execute the function associated with the future. The arguments of the function are bound when the future is created.

Once the function has finished executing, a write operation is performed on the future. The write operation marks the future as completed, and optionally stores data returned by the function. When the result of the delayed computation is being returned.
is needed, a read operation is performed on the future. If the future’s function hasn’t completed when a read operation is performed on it, the reader HPX-thread is suspended until the future is ready. The future facility allows HPX to schedule work early in a program so that when the function value is needed it will already be calculated and available. We use this property in our Fibonacci example below to enable its parallel execution.

**Setup**

The source code for this example can be found here: fibonacci_local.cpp.

To compile this program, go to your HPX build directory (see HPX build system for information on configuring and building HPX) and enter:

```
make examples.quickstart.fibonacci_local
```

To run the program type:

```
./bin/fibonacci_local
```

This should print (time should be approximate):

```
fibonacci(10) == 55
elapsed time: 0.002430 [s]
```

This run used the default settings, which calculate the tenth element of the Fibonacci sequence. To declare which Fibonacci value you want to calculate, use the --n-value option. Additionally you can use the --hpx:threads option to declare how many OS-threads you wish to use when running the program. For instance, running:

```
./bin/fibonacci --n-value 20 --hpx:threads 4
```

Will yield:

```
fibonacci(20) == 6765
elapsed time: 0.062854 [s]
```

**Walkthrough**

Now that you have compiled and run the code, let’s look at how the code works. Since this code is written in C++, we will begin with the main() function. Here you can see that in HPX, main() is only used to initialize the runtime system. It is important to note that application-specific command line options are defined here. HPX uses Boost.Program Options for command line processing. You can see that our programs --n-value option is set by calling the add_options() method on an instance of hpx::program_options::options_description. The default value of the variable is set to 10. This is why when we ran the program for the first time without using the --n-value option the program returned the 10th value of the Fibonacci sequence. The constructor argument of the description is the text that appears when a user uses the --hpx:help option to see what command line options are available. HPX_APPLICATION_STRING is a macro that expands to a string constant containing the name of the HPX application currently being compiled.

In HPX main() is used to initialize the runtime system and pass the command line arguments to the program. If you wish to add command line options to your program you would add them here using the instance of the Boost class options_description, and invoking the public member function .add_options() (see Boost Documentation for more details). hpx::init calls hpx_main() after setting up HPX, which is where the logic of our program is encoded.

---

53 [https://www.boost.org/doc/](https://www.boost.org/doc/)
```cpp
int main(int argc, char* argv[]) {
    // Configure application-specific options
    hpx::program_options::options_description desc_commandline("Usage: " HPX_APPLICATION_STRING " [options]");
    desc_commandline.add_options()
        ("n-value", hpx::program_options::value<std::uint64_t>()->default_value(10), "n value for the Fibonacci function");
    // Initialize and run HPX
    return hpx::init(desc_commandline, argc, argv);
}
```

The `hpx::init` function in `main()` starts the runtime system, and invokes `hpx_main()` as the first HPX-thread. Below we can see that the basic program is simple. The command line option --n-value is read in, a timer (hpx::util::high_resolution_timer) is set up to record the time it takes to do the computation, the `fibonacci` function is invoked synchronously, and the answer is printed out.

```cpp
int hpx_main(hpx::program_options::variables_map& vm) {
    // extract command line argument, i.e. fib(N)
    std::uint64_t n = vm["n-value"].as<std::uint64_t>();

    // Keep track of the time required to execute.
    hpx::util::high_resolution_timer t;
    std::uint64_t r = fibonacci(n);
    char const* fmt = "fibonacci({1}) == {2} elapsed time: {3} [s]"
    hpx::util::format_to(std::cout, fmt, n, r, t.elapsed());

    return hpx::finalize(); // Handles HPX shutdown
}
```

The `fibonacci` function itself is synchronous as the work done inside is asynchronous. To understand what is happening we have to look inside the `fibonacci` function:

```cpp
std::uint64_t fibonacci(std::uint64_t n) {
    if (n < 2)
        return n;
    // Invoking the Fibonacci algorithm twice is inefficient.
    // However, we intentionally demonstrate it this way to create some
    // heavy workload.
    hpx::future<std::uint64_t> n1 = hpx::async(fibonacci, n - 1);
    hpx::future<std::uint64_t> n2 = hpx::async(fibonacci, n - 2);
    return n1.get() + n2.get(); // wait for the Futures to return their values
}
```

This block of code looks similar to regular C++ code. First, if (n < 2), meaning n is 0 or 1, then we re-
turn 0 or 1 (recall the first element of the Fibonacci sequence is 0 and the second is 1). If n is larger than 1 we spawn two new tasks whose results are contained in n1 and n2. This is done using `hpx::async` which takes as arguments a function (function pointer, object or lambda) and the arguments to the function. Instead of returning a `std::uint64_t` like `fibonacci` does, `hpx::async` returns a future of a `std::uint64_t`, i.e. `hpx::future<std::uint64_t>`. Each of these futures represents an asynchronous, recursive call to `fibonacci`. After we’ve created the futures, we wait for both of them to finish computing, we add them together, and return that value as our result. We get the values from the futures using the `get` method. The recursive call tree will continue until n is equal to 0 or 1, at which point the value can be returned because it is implicitly known. When this termination condition is reached, the futures can then be added up, producing the n-th value of the Fibonacci sequence.

Note that calling `get` potentially blocks the calling `HPX`-thread, and lets other `HPX`-threads run in the meantime. There are, however, more efficient ways of doing this. `examples/quickstart/fibonacci_futures.cpp` contains many more variations of locally computing the Fibonacci numbers, where each method makes different tradeoffs in where asynchrony and parallelism is applied. To get started, however, the method above is sufficient and optimizations can be applied once you are more familiar with `HPX`. The example `Dataflow: Interest calculator` presents dataflow, which is a way to more efficiently chain together multiple tasks.

### 2.4.2 Asynchronous execution with `hpx::async` and actions: Fibonacci

This example extends the previous example by introducing actions: functions that can be run remotely. In this example, however, we will still only run the action locally. The mechanism to execute actions stays the same: `hpx::async`. Later examples will demonstrate running actions on remote localities (e.g. `Remote execution with actions: Hello world`).

**Setup**

The source code for this example can be found here: `fibonacci.cpp`.

To compile this program, go to your `HPX` build directory (see `HPX build system` for information on configuring and building `HPX`) and enter:

```
make examples.quickstart.fibonacci
```

To run the program type:

```
./bin/fibonacci
```

This should print (time should be approximate):

```
fibonacci(10) == 55
elapsed time: 0.00186288 [s]
```

This run used the default settings, which calculate the tenth element of the Fibonacci sequence. To declare which Fibonacci value you want to calculate, use the `--n-value` option. Additionally you can use the `--hpx:threads` option to declare how many OS-threads you wish to use when running the program. For instance, running:

```
./bin/fibonacci --n-value 20 --hpx:threads 4
```

Will yield:

```
fibonacci(20) == 6765
elapsed time: 0.233827 [s]
```

**Walkthrough**

The code needed to initialize the `HPX` runtime is the same as in the previous example:
The `hpx::init` function in `main()` starts the runtime system, and invokes `hpx_main()` as the first HPX-thread. The command line option `--n-value` is read in, a timer (`hpx::util::high_resolution_timer`) is set up to record the time it takes to do the computation, the `fibonacci` action is invoked synchronously, and the answer is printed out.

Upon a closer look we see that we've created a `std::uint64_t` to store the result of invoking our `fibonacci_action fib`. This `action` will launch synchronously (as the work done inside of the `action` will be asynchronous itself) and return the result of the Fibonacci sequence. But wait, what is an `action`? And what is this `fibonacci_action`? For starters, an `action` is a wrapper for a function. By wrapping functions, HPX can send packets of work to different processing units. These vehicles allow users to calculate work now, later, or on certain nodes. The first argument to our `action` is the location where the `action` should be run. In this case, we just want to run the `action` on the machine that we are currently on, so we use `hpx::find_here`. To further understand this we turn to the code to find where `fibonacci_action` was defined:

```
//fib_action
// forward declaration of the Fibonacci function
std::uint64_t fibonacci(std::uint64_t n);

// This is to generate the required boilerplate we need for the remote
// invocation to work.
```
A plain action is the most basic form of action. Plain actions wrap simple global functions which are not associated with any particular object (we will discuss other types of actions in Components and actions: Accumulator). In this block of code the function fibonacci() is declared. After the declaration, the function is wrapped in an action in the declaration HPXPLAIN_ACTION. This function takes two arguments: the name of the function that is to be wrapped and the name of the action that you are creating.

This picture should now start making sense. The function fibonacci() is wrapped in an action fibonacci_action, which was run synchronously but created asynchronous work, then returns a std::uint64_t representing the result of the function fibonacci(). Now, let’s look at the function fibonacci():

```cpp
//fib_func
std::uint64_t fibonacci(std::uint64_t n)
{
    if (n < 2)
        return n;
    // We restrict ourselves to execute the Fibonacci function locally.
    hpx::naming::id_type const locality_id = hpx::find_here();

    // Invoking the Fibonacci algorithm twice is inefficient.
    // However, we intentionally demonstrate it this way to create some
    // heavy workload.
    fibonacci_action fib;
    hpx::future<std::uint64_t> n1 =
        hpx::async(fib, locality_id, n - 1);
    hpx::future<std::uint64_t> n2 =
        hpx::async(fib, locality_id, n - 2);
    return n1.get() + n2.get();    // wait for the Futures to return their values
}
```

This block of code is much more straightforward and should look familiar from the previous example. First, if (n < 2), meaning n is 0 or 1, then we return 0 or 1 (recall the first element of the Fibonacci sequence is 0 and the second is 1). If n is larger than 1 we spawn two tasks using hpx::async. Each of these futures represents an asynchronous, recursive call to fibonacci. As previously we wait for both futures to finish computing, get the results, add them together, and return that value as our result. The recursive call tree will continue until n is equal to 0 or 1, at which point the value can be returned because it is implicitly known. When this termination condition is reached, the futures can then be added up, producing the n-th value of the Fibonacci sequence.

2.4.3 Remote execution with actions: Hello world

This program will print out a hello world message on every OS-thread on every locality. The output will look something like this:

```
hello world from OS-thread 1 on locality 0
hello world from OS-thread 1 on locality 1
hello world from OS-thread 0 on locality 0
hello world from OS-thread 0 on locality 1
```

Setup

The source code for this example can be found here: hello_world_distributed.cpp.

To compile this program, go to your HPX build directory (see HPX build system for information on configuring and building HPX) and enter:
make examples.quickstart.hello_world_distributed

To run the program type:

./bin/hello_world_distributed

This should print:

hello world from OS-thread 0 on locality 0

To use more OS-threads use the command line option `--hpx:threads` and type the number of threads that you wish to use. For example, typing:

./bin/hello_world_distributed --hpx:threads 2

will yield:

hello world from OS-thread 1 on locality 0
hello world from OS-thread 0 on locality 0

Notice how the ordering of the two print statements will change with subsequent runs. To run this program on multiple localities please see the section *How to use HPX applications with PBS*.

**Walkthrough**

Now that you have compiled and run the code, let’s look at how the code works, beginning with `main()`:

```cpp
// Here is the main entry point. By using the include 'hpx/hpx_main.hpp' HPX
// will invoke the plain old C-main() as its first HPX thread.
int main()
{
    // Get a list of all available localities.
    std::vector<hpx::naming::id_type> localities =
        hpx::find_all_localities();

    // Reserve storage space for futures, one for each locality.
    std::vector<hpx::lcos::future<void>> futures;
    futures.reserve(localities.size());

    for (hpx::naming::id_type const &node : localities)
    {
        // Asynchronously start a new task. The task is encapsulated in a
        // future, which we can query to determine if the task has
        // completed.
        typedef hello_world_foreman_action action_type;
        futures.push_back(hpx::async<action_type>(node));
    }

    // The non-callback version of hpx::lcos::wait_all takes a single parameter,
    // a vector of futures to wait on. hpx::wait_all only returns when
    // all of the futures have finished.
    hpx::wait_all(futures);

    return 0;
}
```

In this excerpt of the code we again see the use of futures. This time the futures are stored in a vector so that they can easily be accessed. `hpx::wait_all` is a family of functions that wait on for an `std::vector<>` of futures.
to become ready. In this piece of code, we are using the synchronous version of `hpx::wait_all`, which takes one argument (the `std::vector<>` of futures to wait on). This function will not return until all the futures in the vector have been executed.

In *Asynchronous execution with hpx::async and actions: Fibonacci* we used `hpx::find_here` to specify the target of our actions. Here, we instead use `hpx::find_all_localities`, which returns an `std::vector<>` containing the identifiers of all the machines in the system, including the one that we are on.

As in *Asynchronous execution with hpx::async and actions: Fibonacci* our futures are set using `hpx::async<>`. The `hello_world_foreman_action` is declared here:

```cpp
// Define the boilerplate code necessary for the function 'hello_world_foreman'
// to be invoked as an HPX action.
HPX_PLAIN_ACTION(hello_world_foreman, hello_world_foreman_action);
```

Another way of thinking about this wrapping technique is as follows: functions (the work to be done) are wrapped in actions, and actions can be executed locally or remotely (e.g. on another machine participating in the computation).

Now it is time to look at the `hello_world_foreman()` function which was wrapped in the action above:

```cpp
void hello_world_foreman()
{
    // Get the number of worker OS-threads in use by this locality.
    std::size_t const os_threads = hpx::get_os_thread_count();

    // Find the global name of the current locality.
    hpx::naming::id_type const here = hpx::find_here();

    // Populate a set with the OS-thread numbers of all OS-threads on this
    // locality. When the hello world message has been printed on a particular
    // OS-thread, we will remove it from the set.
    std::set<std::size_t> attendance;
    for (std::size_t os_thread = 0; os_thread < os_threads; ++os_thread)
    attendance.insert(os_thread);

    // As long as there are still elements in the set, we must keep scheduling
    // HPX-threads. Because HPX features work-stealing task schedulers, we have
    // no way of enforcing which worker OS-thread will actually execute
    // each HPX-thread.
    while (!attendance.empty())
    {
        // Each iteration, we create a task for each element in the set of
        // OS-threads that have not said "Hello world". Each of these tasks
        // is encapsulated in a future.
        std::vector<hpx::lcos::future<std::size_t>> futures;
        futures.reserve(attendance.size());
        for (std::size_t worker : attendance)
        {
            // Asynchronously start a new task. The task is encapsulated in a
            // future, which we can query to determine if the task has
            // completed. We give the task a hint to run on a particular worker
            // thread, but no guarantees are given by the scheduler that the
            // task will actually run on that worker thread.
            hpx::parallel::execution::default_executor exec{
                hpx::threads::thread_priority_default,
                hpx::threads::thread_stacksize_default,
                hpx::threads::thread_schedule_hint{
                    hpx::threads::thread_schedule_hint_mode_thread, worker});
```
futures.push_back(hpx::async(exec, hello_world_worker, worker));

// Wait for all of the futures to finish. The callback version of the
// hpx::lcos::wait_each function takes two arguments: a vector of futures,
// and a binary callback. The callback takes two arguments; the first
// is the index of the future in the vector, and the second is the
// return value of the future. hpx::lcos::wait_each doesn't return until
// all the futures in the vector have returned.

hpx::lcos::local::spinlock mtx;

hpx::lcos::wait_each(
    hpx::util::unwrapping([&](std::size_t t) {
        if (std::size_t(-1) != t)
        {
            std::lock_guard<hpx::lcos::local::spinlock> lk(mtx);
            attendance.erase(t);
        }
    }),
    futures);

Now, before we discuss hello_world_foreman(), let’s talk about the hpx::wait_each function. The version of hpx::lcos::wait_each invokes a callback function provided by the user, supplying the callback function with the result of the future.

In hello_world_foreman(), an std::set<> called attendance keeps track of which OS-threads have printed out the hello world message. When the OS-thread prints out the statement, the future is marked as ready, and hpx::lcos::wait_each in hello_world_foreman(). If it is not executing on the correct OS-thread, it returns a value of -1, which causes hello_world_foreman() to leave the OS-thread id in attendance.

```cpp
std::size_t hello_world_worker(std::size_t desired)
{
    // Returns the OS-thread number of the worker that is running this
    // HPX-thread.
    std::size_t current = hpx::get_worker_thread_num();
    if (current == desired)
    {
        // The HPX-thread has been run on the desired OS-thread.
        char const* msg = "hello world from OS-thread {1} on locality {2}\n";
        hpx::util::format_to(hpx::cout, msg, desired, hpx::get_locality_id())
        << hpx::flush;

        return desired;
    }

    // This HPX-thread has been run by the wrong OS-thread, make the foreman
    // try again by rescheduling it.
    return std::size_t(-1);
}
```

Because HPX features work stealing task schedulers, there is no way to guarantee that an action will be scheduled on a particular OS-thread. This is why we must use a guess-and-check approach.
2.4.4 Components and actions: Accumulator

The accumulator example demonstrates the use of components. Components are C++ classes that expose methods as a type of HPX action. These actions are called component actions.

Components are globally named, meaning that a component action can be called remotely (e.g. from another machine). There are two accumulator examples in HPX:

In the Asynchronous execution with hpx::async and actions: Fibonacci and the Remote execution with actions: Hello world, we introduced plain actions, which wrapped global functions. The target of a plain action is an identifier which refers to a particular machine involved in the computation. For plain actions, the target is the machine where the action will be executed.

Component actions, however, do not target machines. Instead, they target component instances. The instance may live on the machine that we’ve invoked the component action from, or it may live on another machine.

The component in this example exposes three different functions:

- `reset()` - Resets the accumulator value to 0.
- `add(arg)` - Adds arg to the accumulators value.
- `query()` - Queries the value of the accumulator.

This example creates an instance of the accumulator, and then allows the user to enter commands at a prompt, which subsequently invoke actions on the accumulator instance.

Setup

The source code for this example can be found here: accumulator_client.cpp.

To compile this program, go to your HPX build directory (see HPX build system for information on configuring and building HPX) and enter:

```
make examples.accumulators.accumulator
```

To run the program type:

```
./bin/accumulator_client
```

Once the program starts running, it will print the following prompt and then wait for input. An example session is given below:

```
commands: reset, add [amount], query, help, quit
> add 5
> add 10
> query
15
> add 2
> query
17
> reset
> add 1
> query
1
> quit
```

Walkthrough

Now, let’s take a look at the source code of the accumulator example. This example consists of two parts: an HPX component library (a library that exposes an HPX component) and a client application which uses the library.
This walkthrough will cover the HPX component library. The code for the client application can be found here: accumulator_client.cpp.

An HPX component is represented by two C++ classes:

- **A server class** - The implementation of the components functionality.
- **A client class** - A high-level interface that acts as a proxy for an instance of the component.

Typically, these two classes all have the same name, but the server class usually lives in different sub-namespaces (server). For example, the full names of the two classes in accumulator are:

- `examples::server::accumulator` (server class)
- `examples::accumulator` (client class)

**The server class**

The following code is from: accumulator.hpp.

All HPX component server classes must inherit publicly from the HPX component base class:

```
hpx::components::component_base
```

The accumulator component inherits from `hpx::components::locking_hook`. This allows the runtime system to ensure that all action invocations are serialized. That means that the system ensures that no two actions are invoked at the same time on a given component instance. This makes the component thread safe and no additional locking has to be implemented by the user. Moreover, accumulator component is a component, because it also inherits from `hpx::components::component_base` (the template argument passed to `locking_hook` is used as its base class). The following snippet shows the corresponding code:

```
//accumulator_server_inherit
class accumulator
  : public hpx::components::locking_hook<
```

Our accumulator class will need a data member to store its value in, so let’s declare a data member:

```
//accumulator_server_data_member
```

The constructor for this class simply initializes `value_` to 0:

```
//accumulator_server_ctor
```

Next, let’s look at the three methods of this component that we will be exposing as component actions:

```
//accumulator_methods
    /// Reset the components value to 0.
    void reset()
    {
        // set value_ to 0.
        value_ = 0;
    }

    /// Add the given number to the accumulator.
    void add(argument_type arg)
    {
        // add value_ to arg, and store the result in value_.
        value_ += arg;
    }

    /// Return the current value to the caller.
```

(continues on next page)
argument_type query() const
{
    // Get the value of value_.
    return value_;
Note: The code above must be placed in the global namespace.

The client class

The following code is from accumulator.hpp.

The client class is the primary interface to a component instance. Client classes are used to create components:

```cpp
// Create a component on this locality.
examples::accumulator c = hpx::new_<examples::accumulator>(hpx::find_here());
```

and to invoke component actions:

```cpp
c.add(hpx::launch::apply, 4);
```

Clients, like servers, need to inherit from a base class, this time, hpx::components::client_base:

```cpp
// [accumulator_client_inherit
class accumulator
  : public hpx::components::client_base<
    accumulator, server::accumulator

For readability, we typedef the base class like so:

```cpp
// [accumulator_base_type
typedef hpx::components::client_base<
  accumulator, server::accumulator
```

Here are examples of how to expose actions through a client class:

There are a few different ways of invoking actions:

- **Non-blocking**: For actions which don’t have return types, or when we do not care about the result of an action, we can invoke the action using fire-and-forget semantics. This means that once we have asked HPX to compute the action, we forget about it completely and continue with our computation. We use hpx::apply to invoke an action in a non-blocking fashion.

  ```cpp
  // [accumulator_client_reset_non_blocking
  void reset(hpx::launch::apply_policy)
  {
    HPX_ASSERT(this->get_id());
    typedef server::accumulator::reset_action action_type;
    hpx::apply<action_type>(this->get_id());
  }
  ```

- **Asynchronous**: Futures, as demonstrated in Asynchronous execution with hpx::async: Fibonacci, Asynchronous execution with hpx::async and actions: Fibonacci, and the Remote execution with actions: Hello world, enable asynchronous action invocation. Here’s an example from the accumulator client class:

  ```cpp
  // [accumulator_client_query_async
  hpx::future<argument_type> query(hpx::launch::async_policy)
  {
    HPX_ASSERT(this->get_id());
    typedef server::accumulator::query_action action_type;
    return hpx::async<action_type>(hpx::launch::async, this->get_id());
  }
  ```
• **Synchronous**: To invoke an action in a fully synchronous manner, we can simply call `hpx::async().get()` (e.g., create a future and immediately wait on it to be ready). Here’s an example from the accumulator client class:

```cpp
//accumulator_client_add_sync
void add(argument_type arg)
{
    HPX_ASSERT(this->get_id());

typedef server::accumulator::add_action action_type;
action_type()(this->get_id(), arg);
}
```

Note that `this->get_id()` references a data member of the `hpx::components::client_base` base class which identifies the server accumulator instance.

`hpx::naming::id_type` is a type which represents a global identifier in HPX. This type specifies the target of an action. This is the type that is returned by `hpx::find_here` in which case it represents the *locality* the code is running on.

### 2.4.5 Dataflow: Interest calculator

HPX provides its users with several different tools to simply express parallel concepts. One of these tools is a *local control object* (LCO) called dataflow. An LCO is a type of component that can spawn a new thread when triggered. They are also distinguished from other components by a standard interface which allow users to understand and use them easily. A Dataflow, being an LCO, is triggered when the values it depends on become available. For instance, if you have a calculation $X$ that depends on the results of three other calculations, you could set up a dataflow that would begin the calculation $X$ as soon as the other three calculations have returned their values. Dataflows are set up to depend on other dataflows. It is this property that makes dataflow a powerful parallelization tool. If you understand the dependencies of your calculation, you can devise a simple algorithm which sets up a dependency tree to be executed. In this example, we calculate compound interest. To calculate compound interest, one must calculate the interest made in each compound period, and then add that interest back to the principal before calculating the interest made in the next period. A practical person would of course use the formula for compound interest:

$$F = P(1 + i)^n$$

where $F$ is the future value, $P$ is the principal value, $i$ is the interest rate, and $n$ is the number of compound periods. Nevertheless, we have chosen for the sake of example to manually calculate the future value by iterating:

$$I = Pi$$

and

$$P = P + I$$

**Setup**

The source code for this example can be found here: `interest_calculator.cpp`.

To compile this program, go to your HPX build directory (see *HPX build system* for information on configuring and building HPX) and enter:

```
make examples.quickstart.interest_calculator
```

To run the program type:
This should print:

Final amount: 134.01
Amount made: 34.0096

Walkthrough

Let us begin with main, here we can see that we again are using Boost.Program Options to set our command line variables (see Asynchronous execution with hpx::async and actions: Fibonacci for more details). These options set the principal, rate, compound period, and time. It is important to note that the units of time for \(cp\) and \(time\) must be the same.

```cpp
//interest_main
int main(int argc, char ** argv)
{
    options_description cmdline("Usage: " HPX_APPLICATION_STRING " [options]");

    cmdline.add_options()
        ("principal", value<double>()->default_value(1000), "The principal [\$]")
        ("rate", value<double>()->default_value(7), "The interest rate [%]")
        ("cp", value<int>()->default_value(12), "The compound period [months]"
        ("time", value<int>()->default_value(12*30),
            "The time money is invested [months]"
    );

    return hpx::init(cmdline, argc, argv);
}
```

Next we look at hpx_main.

```cpp
//interest_hpx_main
int hpx_main(variables_map & vm)
{
    using hpx::shared_future;
    using hpx::make_ready_future;
    using hpx::dataflow;
    using hpx::util::unwrapping;
    hpx::naming::id_type here = hpx::find_here();

    double init_principal=vm["principal"].as<double>(); //Initial principal
    double init_rate=vm["rate"].as<double>(); //Interest rate
    int cp=vm["cp"].as<int>(); //Length of a compound period
    int t=vm["time"].as<int>(); //Length of time money is invested

    init_rate/=100; //Rate is a % and must be converted
    t/=cp; //Determine how many times to iterate interest calculation:
    //How many full compound periods can fit in the time invested

    // In non-dataflow terms the implemented algorithm would look like:
    //
    // int t = 5; // number of time periods to use
    // double principal = init_principal;
    // double rate = init_rate;
    //
    // for (int i = 0; i < t; ++i)
```

(continues on next page)
Here we find our command line variables read in, the rate is converted from a percent to a decimal, the number of calculation iterations is determined, and then our shared_futures are set up. Notice that we first place our principal and rate into shares futures by passing the variables init_principal and init_rate using hpx::make_ready_future.

In this way hpx::shared_future<double> principal and rate will be initialized to init_principal and init_rate when hpx::make_ready_future<double> returns a future containing those initial values. These shared futures then enter the for loop and are passed to interest. Next principal and interest are passed to the reassignment of principal using a hpx::dataflow. A dataflow will first wait for its arguments to be ready before launching any callbacks, so add in this case will not begin until both principal and interest are ready. This loop continues for each compound period that must be calculated. To see how interest and principal are calculated in the loop let us look at calc_action and add_action:

// Calculate interest for one period
double calc(double principal, double rate) {
  return principal * rate;
}

// Add the amount made to the principal
double add(double principal, double interest) {
  return principal + interest;
}

After the shared future dependencies have been defined in hpx_main, we see the following statement:

double result = principal.get();
This statement calls `hpx::future::get` on the shared future principal which had its value calculated by our for loop. The program will wait here until the entire dataflow tree has been calculated and the value assigned to result. The program then prints out the final value of the investment and the amount of interest made by subtracting the final value of the investment from the initial value of the investment.

### 2.4.6 Local to remote: 1D stencil

When developers write code they typically begin with a simple serial code and build upon it until all of the required functionality is present. The following set of examples were developed to demonstrate this iterative process of evolving a simple serial program to an efficient, fully-distributed HPX application. For this demonstration, we implemented a 1D heat distribution problem. This calculation simulates the diffusion of heat across a ring from an initialized state to some user-defined point in the future. It does this by breaking each portion of the ring into discrete segments and using the current segment’s temperature and the temperature of the surrounding segments to calculate the temperature of the current segment in the next timestep as shown by Fig. ?? below.

![Heat diffusion example program flow](image)

We parallelize this code over the following eight examples:

- Example 1
- Example 2
- Example 3
- Example 4
- Example 5
- Example 6
- Example 7
- Example 8

The first example is straight serial code. In this code we instantiate a vector `U` that contains two vectors of doubles as seen in the structure `stepper`.

```cpp
//{stepper_1
struct stepper
{
    // Our partition type
    typedef double partition;

    // Our data for one time step
    typedef std::vector<partition> space;

    // Our operator
    static double heat(double left, double middle, double right)
```
}{
    return middle + (k*dt/(dx*dx)) * (left - 2*middle + right);
}

// do all the work on 'nx' data points for 'nt' time steps
space do_work(std::size_t nx, std::size_t nt)
{
    // U[t][i] is the state of position i at time t.
    std::vector<space> U(2);
    for (space& s : U)
        s.resize(nx);

    // Initial conditions: f(0, i) = i
    for (std::size_t i = 0; i != nx; ++i)
        U[0][i] = double(i);

    // Actual time step loop
    for (std::size_t t = 0; t != nt; ++t)
    {
        space const& current = U[t % 2];
        space& next = U[(t + 1) % 2];
        next[0] = heat(current[nx-1], current[0], current[1]);
        for (std::size_t i = 1; i != nx-1; ++i)
            next[i] = heat(current[i-1], current[i], current[i+1]);
        next[nx-1] = heat(current[nx-2], current[nx-1], current[0]);
    }

    // Return the solution at time-step 'nt'.
    return U[nt % 2];
}

Each element in the vector of doubles represents a single grid point. To calculate the change in heat distribution, the temperature of each grid point, along with its neighbors, is passed to the function heat. In order to improve readability, references named current and next are created which, depending on the time step, point to the first and second vector of doubles. The first vector of doubles is initialized with a simple heat ramp. After calling the heat function with the data in the current vector, the results are placed into the next vector.

In example 2 we employ a technique called futurization. Futurization is a method by which we can easily transform a code that is serially executed into a code that creates asynchronous threads. In the simplest case this involves replacing a variable with a future to a variable, a function with a future to a function, and adding a .get() at the point where a value is actually needed. The code below shows how this technique was applied to the struct stepper.

///[stepper_2
struct stepper
{
    // Our partition type
    typedef hpx::shared_future<double> partition;

    // Our data for one time step
    typedef std::vector<partition> space;

    // Our operator
    static double heat(double left, double middle, double right)
In example 2, we redefine our partition type as a shared_future and, in main, create the object result, which is a future to a vector of partitions. We use result to represent the last vector in a string of vectors created for each timestep. In order to move to the next timestep, the values of a partition and its neighbors must be passed to heat once the futures that contain them are ready. In HPX, we have an LCO (Local Control Object) named Dataflow that assists the programmer in expressing this dependency. Dataflow allows us to pass the results of a set of futures to a specified function when the futures are ready. Dataflow takes three types of arguments, one which instructs the dataflow on how to perform the function call (async or sync), the function to call (in this case Op), and futures to the arguments that will be passed to the function. When called, dataflow immediately returns a future to the result of the specified function. This allows users to string dataflows together and construct an execution tree.
After the values of the futures in dataflow are ready, the values must be pulled out of the future container to be passed to the function heat. In order to do this, we use the HPX facility unwrapping, which underneath calls .get() on each of the futures so that the function heat will be passed doubles and not futures to doubles.

By setting up the algorithm this way, the program will be able to execute as quickly as the dependencies of each future are met. Unfortunately, this example runs terribly slow. This increase in execution time is caused by the overheads needed to create a future for each data point. Because the work done within each call to heat is very small, the overhead of creating and scheduling each of the three futures is greater than that of the actual useful work! In order to amortize the overheads of our synchronization techniques, we need to be able to control the amount of work that will be done with each future. We call this amount of work per overhead grain size.

In example 3, we return to our serial code to figure out how to control the grain size of our program. The strategy that we employ is to create “partitions” of data points. The user can define how many partitions are created and how many data points are contained in each partition. This is accomplished by creating the struct partition, which contains a member object data_, a vector of doubles that holds the data points assigned to a particular instance of partition.

In example 4, we take advantage of the partition setup by redefining space to be a vector of shared_futures with each future representing a partition. In this manner, each future represents several data points. Because the user can define how many data points are in each partition, and, therefore, how many data points are represented by one future, a user can control the grainsize of the simulation. The rest of the code is then futurized in the same manner as example 2. It should be noted how strikingly similar example 4 is to example 2.

Example 4 finally shows good results. This code scales equivalently to the OpenMP version. While these results are promising, there are more opportunities to improve the application’s scalability. Currently, this code only runs on one locality, but to get the full benefit of HPX, we need to be able to distribute the work to other machines in a cluster. We begin to add this functionality in example 5.

In order to run on a distributed system, a large amount of boilerplate code must be added. Fortunately, HPX provides us with the concept of a component, which saves us from having to write quite as much code. A component is an object that can be remotely accessed using its global address. Components are made of two parts: a server and a client class. While the client class is not required, abstracting the server behind a client allows us to ensure type safety instead of having to pass around pointers to global objects. Example 5 renames example 4’s struct partition to partition_data and adds serialization support. Next, we add the server side representation of the data in the structure partition_server. Partition_server inherits from hpx::components::component_base, which contains a server-side component boilerplate. The boilerplate code allows a component’s public members to be accessible anywhere on the machine via its Global Identifier (GID). To encapsulate the component, we create a client side helper class. This object allows us to create new instances of our component and access its members without having to know its GID. In addition, we are using the client class to assist us with managing our asynchrony. For example, our client class partition's member function get_data() returns a future to partition_data get_data(). This struct inherits its boilerplate code from hpx::components::client_base.

In the structure stepper, we have also had to make some changes to accommodate a distributed environment. In order to get the data from a particular neighboring partition, which could be remote, we must retrieve the data from all of the neighboring partitions. These retrievals are asynchronous and the function heat_part_data, which, amongst other things, calls heat, should not be called unless the data from the neighboring partitions have arrived. Therefore, it should come as no surprise that we synchronize this operation with another instance of dataflow (found in heat_part). This dataflow receives futures to the data in the current and surrounding partitions by calling get_data() on each respective partition. When these futures are ready, dataflow passes them to the unwrapping function, which extracts the shared_array of doubles and passes them to the lambda. The lambda calls heat_part_data on the locality, which is where the middle partition is.

Although this example could run distributed, it only runs on one locality, as it always uses hpx::find_here() as the target for the functions to run on.

In example 6, we begin to distribute the partition data on different nodes. This is accomplished in stepper::do_work() by passing the GID of the locality where we wish to create the partition to the partition constructor.

2.4. Examples
We distribute the partitions evenly based on the number of localities used, which is described in the function `locidx`. Because some of the data needed to update the partition in `heat_part` could now be on a new `locality`, we must devise a way of moving data to the `locality` of the middle partition. We accomplished this by adding a switch in the function `get_data()` that returns the end element of the buffer `data_` if it is from the left partition or the first element of the buffer if the data is from the right partition. In this way only the necessary elements, not the whole buffer, are exchanged between nodes. The reader should be reminded that this exchange of end elements occurs in the function `get_data()` and, therefore, is executed asynchronously.

Now that we have the code running in distributed, it is time to make some optimizations. The function `heat_part` spends most of its time on two tasks: retrieving remote data and working on the data in the middle partition. Because we know that the data for the middle partition is local, we can overlap the work on the middle partition with that of the possibly remote call of `get_data()`. This algorithmic change, which was implemented in example 7, can be seen below:

```cpp
// The partitioned operator, it invokes the heat operator above on all elements // of a partition.
static partition heat_part(partition const& left,
    partition const& middle, partition const& right)
{
    using hpx::dataflow;
    using hpx::util::unwrapping;

    hpx::shared_future<partition_data> middle_data =
        middle.get_data(partition_server::middle_partition);

    hpx::future<partition_data> next_middle = middle_data.then(
        unwrapping{
            [middle](partition_data const& m) -> partition_data
            {
                HPX_UNUSED(middle);

                // All local operations are performed once the middle data of // the previous time step becomes available.
                std::size_t size = m.size();
                partition_data next(size);
                for (std::size_t i = 1; i != size-1; ++i)
                    next[i] = heat(m[i-1], m[i], m[i+1]);
                return next;
            }
        });

    return dataflow(
        hpx::launch::async,
        unwrapping{
            [left, middle, right](partition_data next, partition_data const& l,
                partition_data const& m, partition_data const& r) -> partition
            {
                HPX_UNUSED(left);
                HPX_UNUSED(right);

                // Calculate the missing boundary elements once the // corresponding data has become available.
            }
        });
}
```

(continues on next page)
Example 8 completes the futurization process and utilizes the full potential of HPX by distributing the program flow to multiple localities, usually defined as nodes in a cluster. It accomplishes this task by running an instance of HPX main on each locality. In order to coordinate the execution of the program, the struct stepper is wrapped into a component. In this way, each locality contains an instance of stepper that executes its own instance of the function do_work(). This scheme does create an interesting synchronization problem that must be solved. When the program flow was being coordinated on the head node, the GID of each component was known. However, when we distribute the program flow, each partition has no notion of the GID of its neighbor if the next partition is on another locality. In order to make the GIDs of neighboring partitions visible to each other, we created two buffers to store the GIDs of the remote neighboring partitions on the left and right respectively. These buffers are filled by sending the GID of newly created edge partitions to the right and left buffers of the neighboring localities.

In order to finish the simulation, the solution vectors named result are then gathered together on locality 0 and added into a vector of spaces overall_result using the HPX functions gather_id and gather_here.

Example 8 completes this example series, which takes the serial code of example 1 and incrementally morphs it into a fully distributed parallel code. This evolution was guided by the simple principles of futurization, the knowledge of grainsize, and utilization of components. Applying these techniques easily facilitates the scalable parallelization of most applications.

2.5 Manual

The manual is your comprehensive guide to HPX. It contains detailed information on how to build and use HPX in different scenarios.

2.5.1 Getting HPX

There are HPX packages available for a few Linux distributions. The easiest way to get started with HPX is to use those packages. We keep an up-to-date list with instructions on the HPX Downloads page. If you use one of the available packages you can skip the next section, HPX build system, but we still recommend that you look through it as it contains useful information on how you can customize HPX at compile-time.

If there isn’t a package available for your platform you should either clone our repository:

or download a package with the source files from HPX Downloads.

54 https://stellar-group.org/downloads/
55 https://stellar-group.org/downloads/
2.5.2 HPX build system

The build system for HPX is based on CMake\(^{56}\). CMake is a cross-platform build-generator tool. CMake does not build the project, it generates the files needed by your build tool (GNU make, Visual Studio, etc.) for building HPX.

This section gives an introduction on how to use our build system to build HPX and how to use HPX in your own projects.

**CMake basics**

CMake is a cross-platform build-generator tool. CMake does not build the project, it generates the files needed by your build tool (gnu make, visual studio, etc.) for building HPX.

In general, the HPX CMake scripts try to adhere to the general CMake policies on how to write CMake-based projects.

**Basic CMake usage**

This section explains basic aspects of CMake, specifically options needed for day-to-day usage.

CMake comes with extensive documentation in the form of html files and on the CMake executable itself. Execute `cmake --help` for further help options.

CMake needs to know which build tool it will generate files for (GNU make, Visual Studio, Xcode, etc.). If not specified on the command line, it will try to guess the build tool based on you environment. Once it has identified the build tool, CMake uses the corresponding generator to create files for your build tool. You can explicitly specify the generator with the command line option `-G "Name of the generator"`. To see the available generators on your platform, execute:

```
cmake --help
```

This will list the generator names at the end of the help text. Generator names are case-sensitive. Example:

```
cmake -G "Visual Studio 9 2008" path/to/hpx
```

For a given development platform there can be more than one adequate generator. If you use Visual Studio "NMake Makefiles" is a generator you can use for building with NMake. By default, CMake chooses the more specific generator supported by your development environment. If you want an alternative generator, you must tell this to CMake with the `-G` option.

**Quick start**

Here, you will use the command-line, non-interactive CMake interface.

1. Download and install CMake here: [CMake Downloads\(^{57}\)](https://www.cmake.org). Version 3.3.2 is the minimally required version for HPX.

2. Open a shell. Your development tools must be reachable from this shell through the `PATH` environment variable.

3. Create a directory for containing the build. Building HPX on the source directory is not supported. cd to this directory:

   ```
   mkdir mybuilddir
   cd mybuilddir
   ```

4. Execute this command on the shell replacing `path/to/hpx/` with the path to the root of your HPX source tree:

```
https://www.cmake.org
```

```
https://www.cmake.org/cmake/resources/software.html
```

56. [https://www.cmake.org](https://www.cmake.org)

57. [https://www.cmake.org/cmake/resources/software.html](https://www.cmake.org/cmake/resources/software.html)
CMake will detect your development environment, perform a series of tests and will generate the files required for building HPX. CMake will use default values for all build parameters. See the *CMake variables used to configure HPX* section for fine-tuning your build.

This can fail if CMake can’t detect your toolset, or if it thinks that the environment is not sane enough. In this case make sure that the toolset that you intend to use is the only one reachable from the shell and that the shell itself is the correct one for your development environment. CMake will refuse to build MinGW makefiles if you have a POSIX shell reachable through the PATH environment variable, for instance. You can force CMake to use various compilers and tools. Please visit CMake Useful Variables for a detailed overview of specific CMake variables.

### Options and variables

Variables customize how the build will be generated. Options are boolean variables, with possible values *ON/OFF*. Options and variables are defined on the CMake command line like this:

```bash
cmake -DVARIABLE=value path/to/hpx
```

You can set a variable after the initial CMake invocation for changing its value. You can also undefine a variable:

```bash
cmake -UVARIABLE path/to/hpx
```

Variables are stored on the CMake cache. This is a file named `CMakeCache.txt` on the root of the build directory. Do not hand-edit it.

Variables are listed here appending its type after a colon. You should write the variable and the type on the CMake command line:

```bash
cmake -DVARIABLE:TYPE=value path/to/llvm/source
```

CMake supports the following variable types: `BOOL` (options), `STRING` (arbitrary string), `PATH` (directory name), `FILEPATH` (file name).

### Prerequisites

#### Supported platforms

At this time, HPX supports the following platforms. Other platforms may work, but we do not test HPX with other platforms, so please be warned.

<table>
<thead>
<tr>
<th>Name</th>
<th>Recommended Version</th>
<th>Minimum Version</th>
<th>Architectures</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux</td>
<td>3.2</td>
<td>2.6</td>
<td>x86-32, x86-64, k1om</td>
</tr>
<tr>
<td>BlueGeneQ</td>
<td>V1R2M0</td>
<td>V1R2M0</td>
<td>PowerPC A2</td>
</tr>
<tr>
<td>Windows</td>
<td>7, Server 2008 R2</td>
<td>Any Windows system</td>
<td>x86-32, x86-64</td>
</tr>
<tr>
<td>Mac OSX</td>
<td></td>
<td>Any OSX system</td>
<td>x86-64</td>
</tr>
</tbody>
</table>

---

Software and libraries

In the simplest case, HPX depends on Boost\textsuperscript{59} and Portable Hardware Locality (HWLOC)\textsuperscript{60}. So, before you read further, please make sure you have a recent version of Boost\textsuperscript{61} installed on your target machine. HPX currently requires at least Boost V1.61.0 to work properly. It may build and run with older versions, but we do not test HPX with those versions, so please be warned.

The installation of Boost is described in detail in Boost’s Getting Started\textsuperscript{62} document. However, if you’ve never used the Boost libraries (or even if you have), here’s a quick primer: Installing Boost.

It is often possible to download the Boost libraries using the package manager of your distribution. Please refer to the corresponding documentation for your system for more information.

In addition, we require a recent version of hwloc in order to support thread pinning and NUMA awareness. See Installing Hwloc for instructions on building Portable Hardware Locality (HWLOC).

HPX is written in 99.99% Standard C++ (the remaining 0.01% is platform specific assembly code). As such, HPX is compilable with almost any standards compliant C++ compiler. A compiler supporting the C++11 Standard is highly recommended. The code base takes advantage of C++11 language features when available (move semantics, rvalue references, magic statics, etc.). This may speed up the execution of your code significantly. We currently support the following C++ compilers: GCC, MSVC, ICPC and clang. For the status of your favorite compiler with HPX visit HPX Buildbot Website\textsuperscript{63}.

<table>
<thead>
<tr>
<th>Name</th>
<th>Recommended version</th>
<th>Minimum version</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compilers</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GNU Compiler Collection (g++)\textsuperscript{64}</td>
<td>4.9 or newer</td>
<td>4.9</td>
<td></td>
</tr>
<tr>
<td>Intel Composer XE Suites\textsuperscript{65}</td>
<td>2014 or newer</td>
<td>2014</td>
<td></td>
</tr>
<tr>
<td>clang: a C language family frontend for LLVM\textsuperscript{66}</td>
<td>3.8 or newer</td>
<td>3.8</td>
<td></td>
</tr>
<tr>
<td>Build System</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMake\textsuperscript{67}</td>
<td>3.9.0</td>
<td>3.3.2</td>
<td>Cuda support 3.9</td>
</tr>
<tr>
<td>Required Libraries</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Boost C++ Libraries\textsuperscript{68}</td>
<td>1.67.0 or newer</td>
<td>1.61.0</td>
<td></td>
</tr>
<tr>
<td>Portable Hardware Locality (HWLOC)\textsuperscript{69}</td>
<td>1.11</td>
<td>1.2 (Xeon Phi: 1.6)</td>
<td></td>
</tr>
</tbody>
</table>

Note: When compiling with the Intel Compiler on Linux systems, we only support C++ Standard Libraries provided by gcc 4.8 and upwards. If the g++ in your path is older than 4.8, please specify the path of a newer g++ by setting CMAKE_CXX_FLAGS=''-gxx-name=/path/to/g++' via CMake\textsuperscript{70}.

\textsuperscript{59} https://www.boost.org/
\textsuperscript{60} https://www.open-mpi.org/projects/hwloc/
\textsuperscript{61} https://www.boost.org/
\textsuperscript{62} https://www.boost.org/doc/libs/1_71_0/more/getting_started/index.html
\textsuperscript{63} http://rostam.cct.lsu.edu/
\textsuperscript{64} https://gcc.gnu.org
\textsuperscript{65} https://software.intel.com/en-us/intel-composer-xe/
\textsuperscript{66} https://clang.llvm.org/
\textsuperscript{67} https://www.cmake.org
\textsuperscript{68} https://www.boost.org/
\textsuperscript{69} https://www.open-mpi.org/projects/hwloc/
\textsuperscript{70} https://www.cmake.org
Note: When building Boost using gcc, please note that it is always a good idea to specify a `cxxflags=-std=c++11` command line argument to `b2 (bjam)`. Note, however, that this is absolutely necessary when using gcc V5.2 and above.

Table 2.3: Software prerequisites for HPX on Windows systems

<table>
<thead>
<tr>
<th>Name</th>
<th>Recommended version</th>
<th>Minimum version</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Compilers</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Visual C++/++ (x64)</td>
<td>2015</td>
<td>2015</td>
<td></td>
</tr>
<tr>
<td><strong>Build System</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMake</td>
<td>3.9.0</td>
<td>3.3.2</td>
<td></td>
</tr>
<tr>
<td><strong>Required Libraries</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Boost</td>
<td>1.67.0 or newer</td>
<td>1.61.0</td>
<td></td>
</tr>
<tr>
<td>Portable Hardware Locality (HWLOC)</td>
<td>1.11</td>
<td>1.5</td>
<td></td>
</tr>
</tbody>
</table>

Table 2.4: Highly recommended optional software prerequisites for HPX on Linux systems

<table>
<thead>
<tr>
<th>Name</th>
<th>Recommended version</th>
<th>Minimum version</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>google-perftools</td>
<td>1.7.1</td>
<td>1.7.1</td>
<td>Used as a replacement for the system allocator, and for allocation diagnostics.</td>
</tr>
<tr>
<td>libunwind</td>
<td>0.99</td>
<td>0.97</td>
<td>Dependency of google-perftools on x86-64, used for stack unwinding.</td>
</tr>
<tr>
<td>Open MPI</td>
<td>1.10.1</td>
<td>1.8.0</td>
<td>Can be used as a highspeed communication library backend for the parcelport.</td>
</tr>
</tbody>
</table>

Note: You need to build the following Boost libraries for HPX: Boost.Filesystem, Boost.ProgramOptions, Boost.Regex, and Boost.System. The following are not needed by default, but are required in certain configurations: Boost.Chrono, Boost.DateTime, Boost.Log, Boost.LogSetup, and Boost.Thread.

Depending on the options you chose while building and installing HPX, you will find that HPX may depend on several other libraries such as those listed below.

Note: In order to use a high speed parcelport, we currently recommend configuring HPX to use MPI so that MPI can be used for communication between different localities. Please set the CMake variable `MPI_CXX_COMPILER` to your MPI C++ compiler wrapper if not detected automatically.

Note: When using OpenMPI please note that Ubuntu (notably 18.04 LTS) and older Debian ship an OpenMPI 2.x built with `--enable-heterogeneous` which may cause communication failures at runtime and should not be used.

---

72 https://www.cmake.org  
73 https://www.boost.org/  
74 https://www.open-mpi.org/projects/hwloc/  
75 https://code.google.com/p/gperftools  
76 https://www.nongnu.org/libunwind  
77 https://www.open-mpi.org
Table 2.5: Optional software prerequisites for HPX on Linux systems

<table>
<thead>
<tr>
<th>Name</th>
<th>Recommended version</th>
<th>Minimum version</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance Application Programming Interface (PAPI)</td>
<td></td>
<td></td>
<td>Used for accessing hardware performance data.</td>
</tr>
<tr>
<td>jemalloc(^{78})</td>
<td>2.1.2</td>
<td>2.1.0</td>
<td>Used as a replacement for the system allocator.</td>
</tr>
<tr>
<td>mi-malloc(^{79})</td>
<td>latest</td>
<td>1.0.0</td>
<td>Used as a replacement for the system allocator.</td>
</tr>
<tr>
<td>Hierarchical Data Format V5 (HDF5)(^{80})</td>
<td>1.8.7</td>
<td>1.6.7</td>
<td>Used for data I/O in some example applications. See important note below.</td>
</tr>
</tbody>
</table>

Table 2.6: Optional software prerequisites for HPX on Windows systems

<table>
<thead>
<tr>
<th>Name</th>
<th>Recommended version</th>
<th>Minimum version</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hierarchical Data Format V5 (HDF5)(^{81})</td>
<td>1.8.7</td>
<td>1.6.7</td>
<td>Used for data I/O in some example applications. See important note below.</td>
</tr>
</tbody>
</table>

**Important:** The C++ HDF5 libraries must be compiled with enabled thread safety support. This has to be explicitly specified while configuring the HDF5 libraries as it is not the default. Additionally, you must set the following environment variables before configuring the HDF5 libraries (this part only needs to be done on Linux):

```
export CFLAGS='"-DHDFdexit"'
export CPPFLAGS='"-DHDFdexit"'
```

**Documentation**

To build the HPX documentation, you need recent versions of the following packages:

- python (2 or 3)
- sphinx (Python package)
- sphinx_rtd_theme (Python package)
- breathe (Python package)
- doxygen

If the Python\(^{82}\) dependencies are not available through your system package manager, you can install them using the Python package manager pip:

```
pip install --user sphinx sphinx_rtd_theme breathe
```

You may need to set the following CMake variables to make sure CMake can find the required dependencies.

---

\(^{78}\) [http://jemalloc.net](http://jemalloc.net)

\(^{79}\) [http://microsoft.github.io/mimalloc/](http://microsoft.github.io/mimalloc/)

\(^{80}\) [https://www.hdfgroup.org/HDF5](https://www.hdfgroup.org/HDF5)

\(^{81}\) [https://www.hdfgroup.org/HDF5](https://www.hdfgroup.org/HDF5)

\(^{82}\) [https://www.python.org](https://www.python.org)
**DOXYGEN_ROOT:PATH**
Specifies where to look for the installation of the Doxygen\(^{83}\) tool.

**SPHINX_ROOT:PATH**
Specifies where to look for the installation of the Sphinx\(^{84}\) tool.

**BREATHE_APIDOC_ROOT:PATH**
Specifies where to look for the installation of the Breathe\(^{85}\) tool.

### Installing Boost

**Important:** When building Boost using gcc, please note that it is always a good idea to specify a `cxxflags=-std=c++11` command line argument to `b2` (bjam). Doing so is absolutely necessary when using gcc V5.2 and above.

**Important:** On Windows, depending on the installed versions of Visual Studio, you might also want to pass the correct toolset to the `b2` command depending on which version of the IDE you want to use. In addition, passing `address-model=64` is highly recommended. It might also be necessary to add command line argument `--build-type=complete` to the `b2` command on the Windows platform.

The easiest way to create a working Boost installation is to compile Boost from sources yourself. This is particularly important as many high performance resources, even if they have Boost installed, usually only provide you with an older version of Boost. We suggest you download the most recent release of the Boost libraries from here: Boost Downloads\(^{86}\). Unpack the downloaded archive into a directory of your choosing. We will refer to this directory as `$BOOST`.

Building and installing the Boost binaries is simple. Regardless of what platform you are on, the basic instructions are as follows (with possible additional platform-dependent command line arguments):

```bash
cd $BOOST
bootstrap --prefix=<where to install boost>
./b2 -j<N>
./b2 install
```

where: `<where to install boost>` is the directory the built binaries will be installed to, and `<N>` is the number of cores to use to build the Boost binaries.

After the above sequence of commands has been executed (this may take a while!), you will need to specify the directory where Boost was installed as `BOOST_ROOT` (<where to install boost>) while executing CMake for HPX as explained in detail in the sections How to install HPX on Unix variants and How to install HPX on Windows.

### Installing Hwloc

**Note:** These instructions are for everything except Windows. On Windows there is no need to build hwloc. Instead, download the latest release, extract the files, and set `HWLOC_ROOT` during CMake configuration to the directory in which you extracted the files.

---

\(^{83}\) [https://www.doxygen.org](https://www.doxygen.org)

\(^{84}\) [http://www.sphinx-doc.org](http://www.sphinx-doc.org)


\(^{86}\) [https://www.boost.org/users/download/](https://www.boost.org/users/download/)
We suggest you download the most recent release of hwloc from here: Hwloc Downloads. Unpack the downloaded archive into a directory of your choosing. We will refer to this directory as $HWLOC.

To build hwloc run:

```
cd $HWLOC
./configure --prefix=<where to install hwloc>
maker -j<N> install
```

where: `<where to install hwloc>` is the directory the built binaries will be installed to, and `<N>` is the number of cores to use to build hwloc.

After the above sequence of commands has been executed, you will need to specify the directory where hwloc was installed as HWLOC_ROOT (`<where to install hwloc>`), while executing CMake for HPX as explained in detail in the sections How to install HPX on Unix variants and How to install HPX on Windows.

Please see Hwloc Documentation for more information about hwloc.

### Building HPX

#### Basic information

Once CMake has been run, the build process can be started. The HPX build process is highly configurable through CMake, and various CMake variables influence the build process. The build process consists of the following parts:

- **The HPX core libraries (target core):** This forms the basic set of HPX libraries. The generated targets are:
  - `hpx`: The core HPX library (always enabled).
  - `hpx_init`: The HPX initialization library that applications need to link against to define the HPX entry points (disabled for static builds).
  - `hpx_wrap`: The HPX static library used to determine the runtime behavior of HPX code and respective entry points for `hpx_main.h`.
  - `iostreams_component`: The component used for (distributed) IO (always enabled).
  - `component_storage_component`: The component needed for migration to persistent storage.
  - `unordered_component`: The component needed for a distributed (partitioned) hash table.
  - `partioned_vector_component`: The component needed for a distributed (partitioned) vector.
  - `memory_component`: A dynamically loaded plugin that exposes memory based performance counters (only available on Linux).
  - `io_counter_component`: A dynamically loaded plugin that exposes I/O performance counters (only available on Linux).
  - `papi_component`: A dynamically loaded plugin that exposes PAPI performance counters (enabled with `HPX_WITH_PAPI:BOOL`, default is Off).

- **HPX Examples (target examples):** This target is enabled by default and builds all HPX examples (disable by setting `HPX_WITH_EXAMPLES:BOOL=Off`). HPX examples are part of the all target and are included in the installation if enabled.

- **HPX Tests (target tests):** This target builds the HPX test suite and is enabled by default (disable by setting `HPX_WITH_TESTS:BOOL =Off`). They are not built by the all target and have to be built separately.

- **HPX Documentation (target docs):** This target builds the documentation, and is not enabled by default (enable by setting `HPX_WITH_DOCUMENTATION:BOOL=On`. For more information see Documentation.

---

87 https://www.open-mpi.org/software/hwloc/v1.11
88 https://www.open-mpi.org/projects/hwloc/doc/
For a complete list of available CMake variables that influence the build of HPX, see CMake variables used to configure HPX.

The variables can be used to refine the recipes that can be found at Platform specific build recipes which show some basic steps on how to build HPX for a specific platform.

In order to use HPX, only the core libraries are required (the ones marked as optional above are truly optional). When building against HPX, the CMake variable HPX_LIBRARIES will contain hpx and hpx_init (for pkgconfig, those are added to the Libs sections). In order to use the optional libraries, you need to specify them as link dependencies in your build (See Creating HPX projects).

As HPX is a modern C++ library, we require a certain minimum set of features from the C++11 standard. In addition, we make use of certain C++14 features if the used compiler supports them. This means that the HPX build system will try to determine the highest support C++ standard flavor and check for availability of those features. That is, the default will be the highest C++ standard version available. If you want to force HPX to use a specific C++ standard version, you can use the following CMake variables:

- HPX_WITH_CXX0X: Enables Pre-C++11 support (This is the minimum required mode on older gcc versions).
- HPX_WITH_CXX11: Enables C++11 support
- HPX_WITH_CXX14: Enables C++14 support
- HPX_WITH_CXX17: Enables C++17 support
- HPX_WITH_CXX2A: Enables (experimental) C++20 support

### Build types

CMake can be configured to generate project files suitable for builds that have enabled debugging support or for an optimized build (without debugging support). The CMake variable used to set the build type is CMAKE_BUILD_TYPE (for more information see the CMake Documentation[^89]). Available build types are:

- **Debug**: Full debug symbols are available as well as additional assertions to help debugging. To enable the debug build type for the HPX API, the C++ Macro HPX_DEBUG is defined.
- **RelWithDebInfo**: Release build with debugging symbols. This is most useful for profiling applications
- **Release**: Release build. This disables assertions and enables default compiler optimizations.
- **RelMinSize**: Release build with optimizations for small binary sizes.

**Important**: We currently don’t guarantee ABI compatibility between Debug and Release builds. Please make sure that applications built against HPX use the same build type as you used to build HPX. For CMake builds, this means that the CMAKE_BUILD_TYPE variables have to match and for projects not using CMake[^90], the HPX_DEBUG macro has to be set in debug mode.

### Platform specific notes

Some platforms require users to have special link and/or compiler flags specified to build HPX. This is handled via CMake’s support for different toolchains (see cmake-toolchains(7)[^91] for more information). This is also used for cross compilation.

HPX ships with a set of toolchains that can be used for compilation of HPX itself and applications depending on HPX. Please see CMake toolchains shipped with HPX for more information.

[^89]: https://cmake.org/cmake/help/latest/variable/CMAKE_BUILD_TYPE.html
[^90]: https://www.cmake.org
In order to enable full static linking with the libraries, the CMake variable \texttt{HPX\_WITH\_STATIC\_LINKING:BOOL} has to be set to \texttt{On}.

\section*{Debugging applications using core files}

For \textit{HPX} to generate useful core files, \textit{HPX} has to be compiled without signal and exception handlers \texttt{HPX\_WITH\_DISABLED\_SIGNAL\_EXCEPTION\_HANDLERS:BOOL}. If this option is not specified, the signal handlers change the application state. For example, after a segmentation fault the stack trace will show the signal handler. Similarly, unhandled exceptions are also caught by these handlers and the stack trace will not point to the location where the unhandled exception was thrown.

In general, core files are a helpful tool to inspect the state of the application at the moment of the crash (post-mortem debugging), without the need of attaching a debugger beforehand. This approach to debugging is especially useful if the error cannot be reliably reproduced, as only a single crashed application run is required to gain potentially helpful information like a stacktrace.

To debug with core files, the operating system first has to be told to actually write them. On most Unix systems this can be done by calling:

\begin{verbatim}
ulimit -c unlimited
\end{verbatim}

in the shell. Now the debugger can be started up with:

\begin{verbatim}
gdb <application> <core file name>
\end{verbatim}

The debugger should now display the last state of the application. The default file name for core files is \textit{core}.

\section*{Platform specific build recipes}

\textbf{Note:} The following build recipes are mostly user-contributed and may be outdated. We always welcome updated and new build recipes.

\section*{How to install \textit{HPX} on Unix variants}

\begin{itemize}
  \item Create a build directory. \textit{HPX} requires an out-of-tree build. This means you will be unable to run CMake in the \textit{HPX} source tree.

  \begin{verbatim}
  cd hpx
  mkdir my_hpx_build
  cd my_hpx_build
  \end{verbatim}

  \item Invoke \texttt{CMake} from your build directory, pointing the \texttt{CMake} driver to the root of your \textit{HPX} source tree.

  \begin{verbatim}
  cmake -DBOOST_ROOT=/root/of/boost/installation
  -DHWLOC_ROOT=/root/of/hwloc/installation
  [other \texttt{CMake} variable definitions] \\
  /path/to/source/tree
  \end{verbatim}

  For instance:

  \begin{verbatim}
  cmake -DBOOST_ROOT=/~packages/boost -DHWLOC_ROOT=/~packages/hwloc -DCMAKE_INSTALL_ \n  --PREFIX=/~packages/hpx ~/downloads/hpx_0.9.10
  \end{verbatim}

  \item Invoke GNU make. If you are on a machine with multiple cores, add the `-jN` flag to your make invocation, where \texttt{N} is the number of parallel processes \textit{HPX} gets compiled with.

\end{itemize}
gmake -j4

Caution: Compiling and linking HPX needs a considerable amount of memory. It is advisable that at least 2 GB of memory per parallel process is available.

Note: Many Linux distributions use make as an alias for gmake.

• To complete the build and install HPX:
  
gmake install

Important: These commands will build and install the essential core components of HPX only. In order to build and run the tests, please invoke:
  
gmake tests && gmake test

and in order to build (and install) all examples invoke:
  
cmake -DHPX_WITH_EXAMPLES=On .
gmake examples

gmake install

For more detailed information about using CMake, please refer to its documentation and also the section Building HPX. Please pay special attention to the section about HPX_WITH_MALLOC:STRING as this is crucial for getting decent performance.

How to install HPX on OS X (Mac)

This section describes how to build HPX for OS X (Mac).

Build (and install) a recent version of Boost, using Clang and libc++

To build Boost with Clang and make it link to libc++ as standard library, you’ll need to set up either of the following in your ~/.user-config.jam file:

```jam
# user-config.jam (put this file into your home directory)
# ...

using clang
:
  : "/usr/bin/clang++"
  : <cxxflags>"-std=c++11 -fcolor-diagnostics"
  <linkflags>"-stdlib=libc++ -L/path/to/libcxx/lib"
;
```

(Again, remember to replace /path/to with whatever you used earlier.)

Then, you can use one of the following for your build command:

```
b2 --build-dir=/tmp/build-boost --layout=versioned toolset=clang install -j4
```
or:

```
b2 --build-dir=/tmp/build-boost --layout=versioned toolset=clang install -j4
```

We verified this using Boost V1.53. If you use a different version, just remember to replace `/usr/local/include/boost-1_53` with whatever prefix you used in your installation.

**Build HPX, finally**

```
cd /path/to
git clone https://github.com/STEllAR-GROUP/hpx.git
mkdir build-hpx && cd build-hpx
```

To build with Clang 3.2, execute:

```
cmake ../hpx
    -DCMAKE_CXX_COMPILER=clang++
    -DBOOST_INCLUDE_DIR=/usr/local/include/boost-1_53
    -DBOOST_LIBRARY_DIR=/usr/local/lib
    -DBOOST_SUFFIX=-clang-darwin32-mt-1_53
make
```

To build with Clang 3.3 (trunk), execute:

```
cmake ../hpx
    -DCMAKE_CXX_COMPILER=clang++
    -DBOOST_INCLUDE_DIR=/usr/local/include/boost-1_53
    -DBOOST_LIBRARY_DIR=/usr/local/lib
    -DBOOST_SUFFIX=-clang-darwin33-mt-1_53
make
```

For more detailed information about using CMake, please refer its documentation and to the section *Building HPX*.

**Alternative installation method of HPX on OS X (Mac)**

Alternatively, you can install a recent version of gcc as well as all required libraries via MacPorts:

1. Install MacPorts

2. Install CMake, gcc 4.8, and hwloc:

```
sudo port install gcc48
sudo port install hwloc
```

You may also want:

```
sudo port install cmake
sudo port install git-core
```

3. Make this version of gcc your default compiler:

```
sudo port install gcc_select
sudo port select gcc mp-gcc48
```

4. Build Boost manually (the Boost package of MacPorts is built with Clang, and unfortunately doesn’t work with a GCC-build version of HPX):
wget https://dl.bintray.com/boostorg/release/1.69.0/source/boost_1_69_0.tar.bz2
tar xjf boost_1_69_0.tar.bz2
pushd boost_1_69_0
export BOOST_ROOT=$HOME/boost_1_69_0
./bootstrap.sh --prefix=$BOOST_DIR
./b2 -j8
./b2 -j8 install
export DYLD_LIBRARY_PATH=$DYLD_LIBRARY_PATH:$BOOST_ROOT/lib
popd

5. Build HPX:

```bash
git clone https://github.com/STEllAR-GROUP/hpx.git
mkdir hpx-build
pushd hpx-build
export HPX_ROOT=$HOME/hpx
cmake -DCMAKE_C_COMPILER=gcc \ 
   -DCMAKE_CXX_COMPILER=g++ \ 
   -DCMAKE_FORTRAN_COMPILER=gfortran \ 
   -DCMAKE_C_FLAGS="-Wno-unused-local-typedefs" \ 
   -DCMAKE_CXX_FLAGS="-Wno-unused-local-typedefs" \ 
   -DBOOST_ROOT=$BOOST_ROOT \ 
   -DHWLOC_ROOT=/opt/local \ 
   -DCMAKE_INSTALL_PREFIX=$HOME/hpx \ 
   $pwd)/../hpx
make -j8
make -j8 install
export DYLD_LIBRARY_PATH=$DYLD_LIBRARY_PATH:$HPX_ROOT/lib/hpx
popd
```

6. Note that you need to set BOOST_ROOT, HPX_ROOT and DYLD_LIBRARY_PATH (for both BOOST_ROOT and HPX_ROOT) every time you configure, build, or run an HPX application.

7. If you want to use HPX with MPI, you need to enable the MPI parcelport, and also specify the location of the MPI wrapper scripts. This can be done using the following command:

```bash
cmake -DHPX_WITH_PARCELPORT_MPI=ON \ 
   -DCMAKE_C_COMPILER=gcc \ 
   -DCMAKE_CXX_COMPILER=g++ \ 
   -DCMAKE_FORTRAN_COMPILER=gfortran \ 
   -DMPI_C_COMPILER=openmpicc \ 
   -DMPI_CXX_COMPILER=openmpic++ \ 
   -DMPI_FORTRAN_COMPILER=openmpif90 \ 
   -DCMAKE_C_FLAGS="-Wno-unused-local-typedefs" \ 
   -DCMAKE_CXX_FLAGS="-Wno-unused-local-typedefs" \ 
   -DBOOST_ROOT=$BOOST_DIR \ 
   -DHWLOC_ROOT=/opt/local \ 
   -DCMAKE_INSTALL_PREFIX=$HOME/hpx \ 
   $pwd)/../hpx
```

How to install HPX on Windows

Installation of required prerequisites

- Download the Boost c++ libraries from Boost Downloads92

92 https://www.boost.org/users/download/
• Install the Boost library as explained in the section Installing Boost

• Install the hwloc library as explained in the section Installing Hwloc

• Download the latest version of CMake binaries, which are located under the platform section of the downloads page at CMake Downloads.93

• Download the latest version of HPX from the STEllAR website: HPX Downloads.94

Installation of the HPX library

• Create a build folder. HPX requires an out-of-tree-build. This means that you will be unable to run CMake in the HPX source folder.

• Open up the CMake GUI. In the input box labelled “Where is the source code:”, enter the full path to the source folder. The source directory is the one where the sources were checked out. CMakeLists.txt files in the source directory as well as the subdirectories describe the build to CMake. In addition to this, there are CMake scripts (usually ending in .cmake) stored in a special CMake directory. CMake does not alter any file in the source directory and doesn’t add new ones either. In the input box labelled “Where to build the binaries:”, enter the full path to the build folder you created before. The build directory is one where all compiler outputs are stored, which includes object files and final executables.

• Add CMake variable definitions (if any) by clicking the “Add Entry” button. There are two required variables you need to define: BOOST_ROOT and HWLOC_ROOT These (PATH) variables need to be set to point to the root folder of your Boost and hwloc installations. It is recommended to set the variable CMAKE_INSTALL_PREFIX as well. This determines where the HPX libraries will be built and installed. If this (PATH) variable is set, it has to refer to the directory where the built HPX files should be installed to.

• Press the “Configure” button. A window will pop up asking you which compilers to use. Select the Visual Studio 10 (64Bit) compiler (it usually is the default if available). The Visual Studio 2012 (64Bit) and Visual Studio 2013 (64Bit) compilers are supported as well. Note that while it is possible to build HPX for x86, we don’t recommend doing so as 32 bit runs are severely restricted by a 32 bit Windows system limitation affecting the number of HPX threads you can create.

• Press “Configure” again. Repeat this step until the “Generate” button becomes clickable (and until no variable definitions are marked in red anymore).

• Press “Generate”.

• Open up the build folder, and double-click hpx.sln.

• Build the INSTALL target.

For more detailed information about using CMake95 please refer its documentation and also the section Building HPX.

How to build HPX under Windows 10 x64 with Visual Studio 2015

• Download the CMake V3.4.3 installer (or latest version) from here.96

• Download the hwloc V1.11.0 (or the latest version) from here and unpack it.

• Download the latest Boost libraries from here and unpack them.

• Build the Boost DLLs and LIBs by using these commands from Command Line (or PowerShell). Open CMD/PowerShell inside the Boost dir and type in:

93 https://www.cmake.org/cmake/resources/software.html
94 https://stellar-group.org/downloads/
95 https://www.cmake.org
96 https://blog.kitware.com/cmake-3-4-3-available-for-download/
97 http://www.open-mpi.org/software/hwloc/v1.11/downloads/hwloc-win64-build-1.11.0.zip
98 https://www.boost.org/users/download/
This batch file will set up everything needed to create a successful build. Now execute:

```
b2.exe link=shared variant=release,debug architecture=x86 address-model=64,threading=multi --build-type=complete install
```

This command will start a (very long) build of all available Boost libraries. Please, be patient.

- Open CMake-GUI.exe and set up your source directory (input field ‘Where is the source code’) to the base directory of the source code you downloaded from HPX’s GitHub pages. Here’s an example of CMake path settings, which point to the Documents/GitHub/hpx folder:

```
<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>HGCOMMAND</td>
<td>C:/Program Files/TortoiseHg/hg.exe native</td>
</tr>
<tr>
<td>HPX_PLATFORM</td>
<td></td>
</tr>
<tr>
<td>HPX_WITH_AGAS_DUMP_REFCNT_E</td>
<td></td>
</tr>
<tr>
<td>HPX_WITH_APEX</td>
<td></td>
</tr>
<tr>
<td>HPX_WITH_AUTOMATIC_SERIALIZ...</td>
<td></td>
</tr>
<tr>
<td>HPX_WITH_AWAIT</td>
<td></td>
</tr>
<tr>
<td>HPX_WITH_BENCHMARK_SCRIPTS_P</td>
<td>C:/Users/Harris/Documents/GitHub/hpx/...</td>
</tr>
<tr>
<td>HPX_WITH_BOOST_ALL_DYNAMIC_I1</td>
<td></td>
</tr>
<tr>
<td>HPX_WITH_COLUMN2_BACKWARD</td>
<td></td>
</tr>
<tr>
<td>HPX_WITH_COMPILER_WARNINGS</td>
<td></td>
</tr>
<tr>
<td>HPX_WITH_COMPRESSION_BZIP2</td>
<td></td>
</tr>
<tr>
<td>HPX_WITH_COMPRESSION_SNAPPY</td>
<td></td>
</tr>
</tbody>
</table>
```

Fig. 2.3: Example CMake path settings.

Inside ‘Where is the source-code’ enter the base directory of your HPX source directory (do not enter the “src” sub-directory!). Inside ‘Where to build the binaries’ you should put in the path where all the building processes will happen. This is important because the building machinery will do an “out-of-tree” build. CMake will not touch or change the original source files in any way. Instead, it will generate Visual Studio Solution Files, which will build HPX packages out of the HPX source tree.

- Set three new environment variables (in CMake, not in Windows environment): BOOST_ROOT, HWLOC_ROOT, CMAKE_INSTALL_PREFIX. The meaning of these variables is as follows:
  - BOOST_ROOT the HPX root directory of the unpacked Boost headers/cpp files.
  - HWLOC_ROOT the HPX root directory of the unpacked Portable Hardware Locality files.
  - CMAKE_INSTALL_PREFIX the HPX root directory where the future builds of HPX should be installed.
**Note:** HPX is a very large software collection, so it is not recommended to use the default C:\Program Files\hpx. Many users may prefer to use simpler paths without whitespace, like C:\bin\hpx or D:\bin\hpx etc.

To insert new env-vars click on “Add Entry” and then insert the name inside “Name”, select PATH as Type and put the path-name in the “Path” text field. Repeat this for the first three variables.

This is how variable insertion will look:

![CMake adding entry](image)

Fig. 2.4: Example CMake adding entry.

Alternatively, users could provide BOOST_LIBRARYDIR instead of BOOST_ROOT: the difference is that BOOST_LIBRARYDIR should point to the subdirectory inside Boost root where all the compiled DLLs/LIBs are. For example,

**BOOST_LIBRARYDIR may point to the bin.v2 subdirectory under the Boost rootdir.** It is important to keep the meanings of these two variables separated from each other:

**BOOST_DIR points to the ROOT folder of the Boost library. BOOST_LIBRARYDIR points to the subdir inside the Boost root folder where the compiled binaries are.**

- Click the ‘Configure’ button of CMake-GUI. You will be immediately presented with a small window where you can select the C++ compiler to be used within Visual Studio. This has been tested using the latest v14 (a.k.a C++ 2015) but older versions should be sufficient too. Make sure to select the 64Bit compiler.

- After the generate process has finished successfully, click the ‘Generate’ button. Now, CMake will put new VS Solution files into the BUILD folder you selected at the beginning.
• Open Visual Studio and load the \texttt{HPX.sln} from your build folder.

• Go to \texttt{CMakePredefinedTargets} and build the \texttt{INSTALL} project:

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{solution_explorer.png}
\caption{Visual Studio INSTALL target.}
\end{figure}

It will take some time to compile everything, and in the end you should see an output similar to this one:

\section*{How to Install HPX on BlueGene/Q}

So far we only support BGClang for compiling HPX on the BlueGene/Q.

• Check if BGClang is available on your installation. If not, obtain and install a copy from the BGClang trac page\footnote{https://trac.alfc.anl.gov/projects/llvm-bgq}.

• Build (and install) a recent version of Hwloc Downloads\footnote{https://www.open-mpi.org/software/hwloc/v1.11}. With the following commands:

\begin{verbatim}
./configure \
--host=powerpc64-bqg-linux \
--prefix=${HOME}/install/hwloc \
--disable-shared \
--enable-static \
CPPFLAGS='-$bgsys/drivers/ppcfloor -I/bgsys/drivers/ppcfloor/spi/include/ \kernel/cnk/'
make
make install
\end{verbatim}

• Build (and install) a recent version of Boost, using BGClang. To build Boost with BGClang, you’ll need to set up the following in your Boost \texttt{~/user-config.jam} file:
# user-config.jam (put this file into your home directory)
using clang
  : bgclang++11
  :

You can then use this as your build command:

```
./bootstrap.sh
./b2 --build-dir=/tmp/build-boost --layout=versioned toolset=clang -j12
```

- Clone the master HPX git repository (or a stable tag):

```
git clone git://github.com/STEllAR-GROUP/hpx.git
```

- Generate the HPX buildfiles using CMake:

```
cmake -DHPX_PLATFORM=BlueGeneQ
  -DCMAKE_TOOLCHAIN_FILE=/path/to/hpx/cmake/toolchains/BGQ.cmake
  -DCMAKE_CXX_COMPILER=bgclang++11
  -DMPI_CXX_COMPILER=mpiclang++11
  -DHWLOC_ROOT=/path/to/hwloc/installation
  -DBOOST_ROOT=/path/to/boost
  -DHPX_WITH_MALLOC=system
  /path/to/hpx
```
• To complete the build and install HPX:

```bash
make -j24
mke install
```

This will build and install the essential core components of HPX only. Use:

```bash
make -j24 examples
make -j24 install
```

to build and install the examples.

**How to Install HPX on the Xeon Phi**

**Installation of the Boost Libraries**

• Download Boost Downloads\(^{101}\) for Linux and unpack the retrieved tarball.

• Adapt your `~/user-config.jam` to contain the following lines:

```bash
## Toolset to be used for compiling for the host
using intel
  : host
  :
  : <cxxflags>"-std=c++0x"
 ;

## Toolset to be used for compiling for the Xeon Phi
using intel
  : mic
  :
  : <cxxflags>"-std=c++0x -mmic"
   <linkflags>"-std=c++0x -mmic"
 ;
```

• Change to the directory you unpacked Boost in (which will be referred to as `$BOOST_ROOT` from now on) and execute the following commands:

```bash
./bootstrap.sh
./b2 toolset=intel-mic -j<N>
```

You should now have all the required Boost libraries.

**Installation of the Hwloc library**

• Download Hwloc Downloads\(^{102}\), unpack the retrieved tarball and change to the newly created directory.

• Run the configure-make-install procedure as follows:

```bash
CC=icc CFLAGS=-mmic CXX=icpc CXXFLAGS=-mmic LDFLAGS=-mmic ./configure --host=x86_64-klm-linux --prefix=$HWLOC_ROOT
make
make install
```

\(^{101}\) https://www.boost.org/users/download/

\(^{102}\) https://www.open-mpi.org/software/hwloc/v1.11
**Important:** The minimally required version of the Portable Hardware Locality (HWLOC) library on the Intel Xeon Phi is V1.6.

You now have a working hwloc installation in $HWLOC_ROOT.

**Building HPX**

After all the prerequisites have been successfully installed, we can now start building and installing HPX. The build procedure is almost the same as the one for *How to install HPX on Unix variants* with the sole difference that you have to enable the Xeon Phi in the CMake Build system. This is achieved by invoking CMake in the following way:

```bash
cmake
-DCMAKE_TOOLCHAIN_FILE=/path/to/hpx/cmake/toolchains/XeonPhi.cmake
-DBOOST_ROOT=$BOOST_ROOT
-DHWLOC_ROOT=$HWLOC_ROOT
/path/to/hpx
```

For more detailed information about using CMake, please refer to its documentation and to the section *Building HPX*. Please pay special attention to the section about `HPX_WITH_MALLOC:STRING` as this is crucial for getting decent performance on the Xeon Phi.

**How to install HPX on Fedora distributions**

**Important:** There are official HPX packages for Fedora. Unless you want to customize your build you may want to start off with the official packages. Instructions can be found on the HPX Downloads page.

**Note:** This section of the manual is based off of our collaborator Patrick Diehl’s blog post [Installing hpx on Fedora](http://diehlpk.github.io/2015/08/04/hpx-fedora.html).

- Install all packages for minimal installation:

  ```bash
  sudo dnf install gcc-c++ cmake boost-build boost boost-devel hwloc-devel
  hwloc gcc-gfortran papi-devel gperftools-devel docbook-dtds
  docbook-style-xsl libsodium-devel doxygen boost-doc hdf5-devel
  fop boost-devel boost-openmpi-devel boost-mpich-devel
  ```

- Get the development branch of HPX:

  ```bash
git clone https://github.com/STEllAR-GROUP/hpx.git
```

- Configure it with CMake:

  ```bash
cd hpx
mkdir build
cd build
cmake -DCMAKE_INSTALL_PREFIX=/opt/hpx ..
make -j
make install
```

---

103 [https://stellar-group.org/downloads/](https://stellar-group.org/downloads/)
104 [http://diehlpk.github.io/2015/08/04/hpx-fedora.html](http://diehlpk.github.io/2015/08/04/hpx-fedora.html)
**Note:** To build HPX without examples use:

```bash
cmake -DCMAKE_INSTALL_PREFIX=/opt/hpx -DHPX_WITH_EXAMPLES=Off ..
```

- Add the library path of HPX to ldconfig:

```bash
sudo echo /opt/hpx/lib > /etc/ld.so.conf.d/hpx.conf
sudo ldconfig
```

**How to install HPX on Arch distributions**

**Important:** There are HPX packages for Arch in the AUR. Unless you want to customize your build, you may want to start off with those. Instructions can be found on the [HPX Downloads](https://stellar-group.org/downloads/) page.

- Install all packages for a minimal installation:

```bash
sudo pacman -S gcc clang cmake boost hwloc gperftools
```

- For building the documentation, you will need to further install the following:

```bash
sudo pacman -S doxygen python-pip

pip install --user sphinx sphinx_rtd_theme breathe
```

The rest of the installation steps are the same as those for the Fedora or Unix variants.

**How to install HPX on Debian-based distributions**

- Install all packages for a minimal installation:

```bash
sudo apt install cmake libboost-all-dev hwloc libgoogle-perftools-dev
```

- To build the documentation you will need to further install the following:

```bash
sudo apt install doxygen python-pip

pip install --user sphinx sphinx_rtd_theme breathe
```

or the following if you prefer to get Python packages from the Debian repositories:

```bash
sudo apt install doxygen python-sphinx python-sphinx-rtd-theme python-breathe
```

The rest of the installation steps are same as those for the Fedora or Unix variants.

**CMake toolchains shipped with HPX**

In order to compile HPX for various platforms, we provide a variety of toolchain files that take care of setting up various CMake variables like compilers, etc. They are located in the `cmake/toolchains` directory:

- ARM-gcc
- BGION-gcc

---

105 https://stellar-group.org/downloads/
• **BGQ**
• **Cray**
• **CrayKNL**
• **CrayKNLStatic**
• **CrayStatic**
• **XeonPhi**

To use them, pass the `-DCMAKE_TOOLCHAIN_FILE=<toolchain>` argument to the CMake invocation.

### ARM-gcc

```cpp
# Copyright (c) 2015 Thomas Heller
#
# SPDX-License-Identifier: BSL-1.0
# Distributed under the Boost Software License, Version 1.0. (See accompanying
# file LICENSE_1_0.txt or copy at http://www.boost.org/LICENSE_1_0.txt)
set(CMAKE_SYSTEM_NAME Linux)
set(CMAKE_CROSSCOMPILING ON)
# Set the gcc Compiler
set(CMAKE_CXX_COMPILER arm-linux-gnueabihf-g++-4.8)
set(CMAKE_C_COMPILER arm-linux-gnueabihf-gcc-4.8)
set(HPX_WITH_GENERIC_CONTEXT_COROUTINES ON CACHE BOOL "enable generic coroutines")
set(CMAKE_FIND_ROOT_PATH_MODE_PROGRAM NEVER)
set(CMAKE_FIND_ROOT_PATH_MODE_LIBRARY ONLY)
set(CMAKE_FIND_ROOT_PATH_MODE_INCLUDE ONLY)
set(CMAKE_FIND_ROOT_PATH_MODE_PACKAGE ONLY)
```

### BGION-gcc

```cpp
# Copyright (c) 2014 John Biddiscombe
#
# SPDX-License-Identifier: BSL-1.0
# Distributed under the Boost Software License, Version 1.0. (See accompanying
# file LICENSE_1_0.txt or copy at http://www.boost.org/LICENSE_1_0.txt)
# This is the default toolchain file to be used with CNK on a BlueGene/Q. It sets
# the appropriate compile flags and compiler such that HPX will compile.
# Note that you still need to provide Boost, hwloc and other utility libraries
# like a custom allocator yourself.
#
# Usage : cmake -DCMAKE_TOOLCHAIN_FILE=~/src/hpx/cmake/toolchains/BGION-gcc.cmake ~/src/hpx
#
set(CMAKE_SYSTEM_NAME Linux)
# Set the gcc Compiler
set(CMAKE_CXX_COMPILER g++)
set(CMAKE_C_COMPILER gcc)
#set(CMAKE_Fortran_COMPILER)
# Add flags we need for BGAS compilation
set(CMAKE_CXX_FLAGS_INIT
"-D__powerpc__ -D__bgion__ -I/gpfs/bbp.cscs.ch/home/biddisco/src/bgas/rdmahelper 
   -I/home/biddisco/src/bgas/rdmahelper"
  CACHE STRING "Initial compiler flags used to compile for BGAS"
)
```

(continues on next page)
#-DHPX_SMALL_STACK_SIZE=0x200000 -DHPX_MEDIUM_STACK_SIZE=0x200000 -DHPX_LARGE_STACK_SIZE=0x200000 -DHPX_HUGE_STACK_SIZE=0x200000

set(CMAKE_EXE_LINKER_FLAGS_INIT "-L/gpfs/bbp.cscs.ch/apps/bgas/tools/gcc/gcc-4.8.2/install/lib64 -latomic -lrt" CACHE STRING "BGAS flags")
set(CMAKE_C_FLAGS_INIT "-D__powerpc__ -I/gpfs/bbp.cscs.ch/home/biddisco/src/bgas/rdmahelper" CACHE STRING "BGAS flags")

# We do not perform cross compilation here ... 
set(CMAKE_CROSSCOMPILING OFF)
set(HPX_PLATFORM "native")

# Enable generic coroutines (and use posix version) 
set(HPX_WITH_GENERIC_CONTEXT_COROUTINES OFF CACHE BOOL "disable generic coroutines")
# BGAS nodes support ibverbs, but it is deprecated 
set(HPX_WITH_PARCELPORT_VERBS OFF CACHE BOOL "")
# Always disable the tcp parcelport as it is non-functional on the BGQ. 
set(HPX_WITH_PARCELPORT_TCP OFF CACHE BOOL "")
set(HPX_WITH_PARCELPORT_MPI OFF CACHE BOOL "")
set(HPX_WITH_MAX_CPU_COUNT "64" CACHE STRING "")
if(NOT DEFINED HPX_WITH_MALLOC)
set(HPX_WITH_MALLOC "system" CACHE STRING "")
endif()
set(HPX_HIDDEN_VISIBILITY OFF CACHE BOOL "")

# Convenience setup for jb @ bbpbg2.cscs.ch 
set(BOOST_ROOT "/gpfs/bbp.cscs.ch/home/biddisco/apps/gcc-4.8.2/boost_1_56_0")
set(HWLOC_ROOT "/gpfs/bbp.cscs.ch/home/biddisco/apps/gcc-4.8.2/hwloc-1.8.1")
set(CMAKE_BUILD_TYPE "Debug" CACHE STRING "Default build")

# # Testing flags 
#
set(BUILD_TESTING ON CACHE BOOL "Testing enabled by default")
set(HPX_WITH_TESTS ON CACHE BOOL "Testing enabled by default")
set(HPX_WITH_TESTS_BENCHMARKS ON CACHE BOOL "Testing enabled by default")
set(HPX_WITH_TESTS_REGRESSIONS ON CACHE BOOL "Testing enabled by default")
set(HPX_WITH_TESTS_UNIT ON CACHE BOOL "Testing enabled by default")
set(HPX_WITH_TESTS_EXAMPLES OFF CACHE BOOL "Testing enabled by default")
set(HPX_WITH_TESTS_EXTERNAL_BUILD OFF CACHE BOOL "Turn off build of cmake build tests")
set(DART_TESTING_TIMEOUT 45 CACHE STRING "Life is too short")
set(DART_TESTING_TIMEOUT 45 CACHE STRING "Life is too short")

BGQ

# Copyright (c) 2014 Thomas Heller 
#
# SPDX-License-Identifier: BSL-1.0 
# Distributed under the Boost Software License, Version 1.0. (See accompanying 
# file LICENSE_1_0.txt or copy at http://www.boost.org/LICENSE_1_0.txt) 
# 
# This is the default toolchain file to be used with CNK on a BlueGene/Q. It sets 
# the appropriate compile flags and compiler such that HPX will compile.
# Note that you still need to provide Boost, hwloc and other utility libraries
# like a custom allocator yourself.

set(CMAKE_SYSTEM_NAME Linux)
set(CMAKE_CXX_COMPILER bgclang++11)
set(CMAKE_C_COMPILER bgclang)
set(MPI_CXX_COMPILER mpgclang++11)
set(MPI_C_COMPILER mpgclang)

set(CMAKE_C_FLAGS_INIT "")
set(CMAKE_C_COMPILE_OBJECT "${CMAKE_C_COMPILER} -fPIC <DEFINES> <FLAGS> -o <OBJECT> -c " <SOURCE>" CACHE STRING ")
set(CMAKE_C_LINK_EXECUTABLE "${CMAKE_C_COMPILER} -fPIC -dynamic <FLAGS> <CMAKE_C_LINK_FLAGS> <OBJECTS> -o <TARGET> <LINK_LIBRARIES>" CACHE STRING ")
set(CMAKE_C_CREATE_SHARED_LIBRARY "${CMAKE_C_COMPILER} -fPIC -shared <CMAKE_SHARED_LIBRARY_CXX_FLAGS> <LANGUAGE_COMPILE_FLAGS> <LINK_FLAGS> <CMAKE_SHARED_LIBRARY_CREATE_CXX_FLAGS> <SONAME_FLAG> <TARGET_SONAME> -o <TARGET> <OBJECTS> <LINK_LIBRARIES>" CACHE STRING ")

set(CMAKE_CXX_FLAGS_INIT "")
set(CMAKE_CXX_COMPILE_OBJECT "${CMAKE_CXX_COMPILER} -fPIC <DEFINES> <FLAGS> -o <OBJECT> -c " <SOURCE>" CACHE STRING ")
set(CMAKE_CXX_LINK_EXECUTABLE "${CMAKE_CXX_COMPILER} -fPIC -dynamic <FLAGS> <CMAKE_CXX_LINK_FLAGS> <OBJECTS> -o <TARGET> <LINK_LIBRARIES>" CACHE STRING ")
set(CMAKE_CXX_CREATE_SHARED_LIBRARY "${CMAKE_CXX_COMPILER} -fPIC -shared <CMAKE_SHARED_LIBRARY_CXX_FLAGS> <LANGUAGE_COMPILE_FLAGS> <LINK_FLAGS> <CMAKE_SHARED_LIBRARY_CREATE_CXX_FLAGS> <SONAME_FLAG> <TARGET_SONAME> -o <TARGET> <OBJECTS> <LINK_LIBRARIES>" CACHE STRING ")

set(CMAKE_Fortran_FLAGS_INIT "")
set(CMAKE_Fortran_COMPILE_OBJECT "${CMAKE_FortranCompiler} -fPIC <DEFINES> <FLAGS> -o <OBJECT> -c " <SOURCE>" CACHE STRING ")
set(CMAKE_Fortran_LINK_EXECUTABLE "${CMAKE_FortranCompiler} -fPIC -dynamic <FLAGS> <CMAKE_Fortran_LINK_FLAGS> <OBJECTS> -o <TARGET> <LINK_LIBRARIES>" CACHE STRING ")
set(CMAKE_Fortran_CREATE_SHARED_LIBRARY "${CMAKE_FortranCompiler} -fPIC -shared <CMAKE_SHARED_LIBRARY_Fortran_FLAGS> <LANGUAGE_COMPILE_FLAGS> <LINK_FLAGS> <CMAKE_SHARED_LIBRARY_CREATE_Fortran_FLAGS> <SONAME_FLAG> <TARGET_SONAME> -o <TARGET> <OBJECTS> <LINK_LIBRARIES>" CACHE STRING ")

set(CMAKE_FIND_ROOT_PATH_MODE_PROGRAM BOTH)
set(CMAKE_FIND_ROOT_PATH_MODE_LIBRARY ONLY)
set(CMAKE_FIND_ROOT_PATH_MODE_INCLUDE ONLY)
set(CMAKE_FIND_ROOT_PATH_MODE_PACKAGE ONLY)

set(HPX_PLATFORM "BlueGeneQ")
set(HPX_WITH_PARCELPORT_VERBS OFF)
set(HPX_WITH_PARCELPORT_TCP OFF)
set(HPX_WITH_MPI_PARCELPORT OFF)
set(HPX_WITH_MAX_CPU_COUNT "64")
# We default to ttbmalloc as our allocator on the MIC
if(NOT DEFINED HPX_WITH_MALLOC)
    set(HPX_WITH_MALLOC "system" CACHE STRING "")
endif()

Cray

# Copyright (c) 2014 Thomas Heller
#
# SPDX-License-Identifier: BSL-1.0
# Distributed under the Boost Software License, Version 1.0. (See accompanying
# file LICENSE_1_0.txt or copy at http://www.boost.org/LICENSE_1_0.txt)
#
# This is the default toolchain file to be used with Intel Xeon PHIs. It sets
# the appropriate compile flags and compiler such that HPX will compile.
# Note that you still need to provide Boost, hwloc and other utility libraries
# like a custom allocator yourself.
#
# set(CMAKE_SYSTEM_NAME Cray-CNK-Intel)
if(HPX_WITH_STATIC_LINKING)
    set_property(GLOBAL PROPERTY TARGET_SUPPORTS_SHARED_LIBS FALSE)
else()
endif()
# Set the Cray Compiler Wrapper
set(CMAKE_CXX_COMPILER CC)
set(CMAKE_C_COMPILER cc)
set(CMAKE_Fortran_COMPILER ftn)
if (CMAKE_VERSION VERSION_GREATER 3.3.9)
    set(__includes "<INCLUDES>"
else()
endif()
set(CMAKE_C_FLAGS_INIT "") CACHE STRING ""
set(CMAKE_SHARED_LIBRARY_C_FLAGS "-fPIC -shared" CACHE STRING "")
set(CMAKE_SHARED_LIBRARY_CREATE_C_FLAGS "-fPIC -shared" CACHE STRING "")
set(CMAKE_C_COMPILER_OBJECT "<CMAKE_C_COMPILER> -shared -fPIC <DEFINES> ${__includes}
˓→<FLAGS> -o <OBJECT> -c <SOURCE>" CACHE STRING "")
set(CMAKE_C_LINK_EXECUTABLE "<CMAKE_C_COMPILER> -fPIC -dynamic <FLAGS> <CMAKE_C_LINK_
˓→<FLAGS> <OBJECTS> -o <TARGET> <LINK_LIBRARIES>" CACHE STRING "")
set(CMAKE_C_CREATE_SHARED_LIBRARY "<CMAKE_C_COMPILER> -fPIC -shared <CMAKE_SHARED_
˓→LIBRARY_CXX_FLAGS> <LANGUAGE_COMPILE_FLAGS> <LINK_FLAGS> <CMAKE_SHARED_LIBRARY_
˓→CREATE_CXX_FLAGS> <SONAME_FLAG> <TARGET_SONAME> -o <TARGET> <OBJECTS> <LINK_
˓→LIBRARIES> " CACHE STRING "")
set(CMAKE_CXX_FLAGS_INIT "") CACHE STRING ""
set(CMAKE_SHARED_LIBRARY_CXX_FLAGS "-fPIC -shared" CACHE STRING "")
set(CMAKE_SHARED_LIBRARY_CREATE_CXX_FLAGS "-fPIC -shared" CACHE STRING "")
set(CMAKE_C_COMPILER_OBJECT "<CMAKE_C_COMPILER> -shared -fPIC <DEFINES> ${__
˓→includes} <FLAGS> -o <OBJECT> -c <SOURCE>" CACHE STRING "")
set(CMAKE_CXX_LINK_EXECUTABLE "<CMAKE_CXX_COMPILER> -fPIC -dynamic <FLAGS> <CMAKE_CXX_
˓→<FLAGS> <OBJECTS> -o <TARGET> <LINK_LIBRARIES>" CACHE STRING "")
set(CMAKE_CXX_CREATE_SHARED_LIBRARY "<CMAKE_CXX_COMPILER> -fPIC -shared <CMAKE_SHARED_
˓→LIBRARY_CXX_FLAGS> <LANGUAGE_COMPILE_FLAGS> <LINK_FLAGS> <CMAKE_SHARED_LIBRARY_
˓→CREATE_CXX_FLAGS> <SONAME_FLAG> <TARGET_SONAME> -o <TARGET> <OBJECTS> <LINK_
˓→LIBRARIES> " CACHE STRING "")
set(CMAKE_Fortran_FLAGS_INIT "") CACHE STRING ""
set(CMAKE_SHARED_LIBRARY_Fortran_FLAGS "-fPIC" CACHE STRING "")
set(CMAKE_SHARED_LIBRARY_CREATE_Fortran_FLAGS "-shared" CACHE STRING "")
set(CMAKE_Fortran_COMPILE_OBJECT "<CMAKE_Fortran_COMPILER> -shared -fPIC \"DEFINES\" ${_includes} <FLAGS> -o <OBJECT> -c <SOURCE>" CACHE STRING \"")
set(CMAKE_Fortran_LINK_EXECUTABLE "<CMAKE_Fortran_COMPILER> -fPIC -dynamic <FLAGS> <CMAKE_Fortran_LINK_FLAGS> <LINK_FLAGS> <OBJECTS> -o <TARGET> <LINK_LIBRARIES>")
set(CMAKE_Fortran_CREATE_SHARED_LIBRARY "<CMAKE_Fortran_COMPILER> -fPIC <LANGUAGE_COMPILE_FLAGS> <CMAKE_SHARED_LIBRARY_Fortran_FLAGS> <CMAKE_SHARED_LIBRARY_CREATE_Fortran_FLAGS> <SONAME_FLAG><TARGET_SONAME> -o <TARGET> <OBJECTS> <LINK_LIBRARIES> \"CACHE STRING \"")
# Disable searches in the default system paths. We are cross compiling after all
# and cmake might pick up wrong libraries that way
set(CMAKE_FIND_ROOT_PATH_MODE_PROGRAM BOTH)
set(CMAKE_FIND_ROOT_PATH_MODE_LIBRARY ONLY)
set(CMAKE_FIND_ROOT_PATH_MODE_INCLUDE ONLY)
set(CMAKE_FIND_ROOT_PATH_MODE_PACKAGE ONLY)
set(HPX_WITH_PARCELPORT_TCP ON CACHE BOOL \"")
set(HPX_WITH_PARCELPORT_MPI ON CACHE BOOL \"")
set(HPX_WITH_PARCELPORT_MPI_MULTITHREADED OFF CACHE BOOL \"")
set(HPX_WITH_PARCELPORT_LIBFABRIC ON CACHE BOOL \"")
set(HPX_PARCELPORT_LIBFABRIC_PROVIDER "gni" CACHE STRING \"
See libfabric docs for details, gni,verbs,psm2 etc etc")
set(HPX_PARCELPORT_LIBFABRIC_THROTTLE_SENDS "256" CACHE STRING \"
Max number of messages in flight at once")
set(HPX_PARCELPORT_LIBFABRIC_WITH_DEV_MODE OFF CACHE BOOL \"
Custom libfabric logging flag")
set(HPX_PARCELPORT_LIBFABRIC_WITH_LOGGING OFF CACHE BOOL \"
Libfabric parcelport logging on/off flag")
set(HPX_WITH_ZERO_COPY_SERIALIZATION_THRESHOLD "4096" CACHE STRING \"
The threshold in bytes to when perform zero copy optimizations (default: 128)")
# We do a cross compilation here ...
set(CMAKE_CROSSCOMPILING ON CACHE BOOL \")

CrayKNL

# Copyright (c) 2014 Thomas Heller
#
# SPDX-License-Identifier: BSL-1.0
# Distributed under the Boost Software License, Version 1.0. (See accompanying
# file LICENSE_1_0.txt or copy at http://www.boost.org/LICENSE_1_0.txt)
#
# This is the default toolchain file to be used with Intel Xeon PHIs. It sets
# the appropriate compile flags and compiler such that HPX will compile.
# Note that you still need to provide Boost, hwloc and other utility libraries
# like a custom allocator yourself.
#
if(HPX_WITH_STATIC_LINKING)
  set_property(GLOBAL PROPERTY TARGET_SUPPORTS_SHARED_LIBS FALSE)
else()
endif()
# Set the Cray Compiler Wrapper
set(CMAKE_CXX_COMPILER CC)
set(CMAKE_C_COMPILER cc)
set(CMAKE_Fortran_COMPILER ftn)
if (CMAKE_VERSION VERSION_GREATER 3.3.9)
  set(_includes "<INCLUDES>")
endif()
set(CMAKE_FLAGS_INIT "" CACHE STRING \")
set(CMAKE_SHARED_LIBRARY_C_FLAGS "-fPIC -shared" CACHE STRING "")
set(CMAKE_SHARED_LIBRARY_CREATE_C_FLAGS "-fPIC -shared" CACHE STRING "")
set(CMAKE_C_COMPILE_OBJECT "<CMAKE_C_COMPILER> -shared -fPIC <DEFINES> ${__includes} \"FLAGS\" -o <OBJECT> -c <SOURCE>" CACHE STRING "")
set(CMAKE_C_LINK_EXECUTABLE "<CMAKE_C_COMPILER> -fPIC <FLAGS> <CMAKE_C_LINK_FLAGS> \"OBJECTS\" -o <TARGET> \"LINK_LIBRARIES\"" CACHE STRING "")
set(CMAKE_C_CREATE_SHARED_LIBRARY "<CMAKE_C_COMPILER> -fPIC -shared <CMAKE_SHARED_LIBRARY_CXX_FLAGS> <LANGUAGE_COMPILE_FLAGS> <LINK_FLAGS> <CMAKE_SHARED_LIBRARY_CREATE_CXX_FLAGS> \"CREATE_CXX_FLAGS\" <SONAME_FLAG><TARGET_SONAME> -o <TARGET> <OBJECTS> <LINK_LIBRARIES>" CACHE STRING "")
#
set(CMAKE_CXX_FLAGS_INIT "" CACHE STRING "")
set(CMAKE_SHARED_LIBRARY_CXX_FLAGS "-fPIC -shared" CACHE STRING "")
set(CMAKE_SHARED_LIBRARY_CREATE_CXX_FLAGS "-fPIC -shared" CACHE STRING "")
set(CMAKE_CXX_COMPILE_OBJECT "<CMAKE_CXX_COMPILER> -shared -fPIC <DEFINES> ${__includes} <FLAGS> -o <OBJECT> -c <SOURCE>" CACHE STRING "")
set(CMAKE_CXX_LINK_EXECUTABLE "<CMAKE_CXX_COMPILER> -fPIC -dynamic <FLAGS> <CMAKE_CXX_LINK_FLAGS> <LINK_FLAGS> <OBJECTS> -o <TARGET> <LINK_LIBRARIES>" CACHE STRING "")
set(CMAKE_CXX_CREATE_SHARED_LIBRARY "<CMAKE_CXX_COMPILER> -fPIC -shared <CMAKE_SHARED_LIBRARY_CXX_FLAGS> <LANGUAGE_COMPILE_FLAGS> <LINK_FLAGS> <CMAKE_SHARED_LIBRARY_CREATE_CXX_FLAGS> \"CREATE_CXX_FLAGS\" <SONAME_FLAG><TARGET_SONAME> -o <TARGET> <OBJECTS> <LINK_LIBRARIES>" CACHE STRING "")
#
# Disable searches in the default system paths. We are cross compiling after all
# and cmake might pick up wrong libraries that way
set(CMAKE_FIND_ROOT_PATH_MODE_PROGRAM BOTH)
set(CMAKE_FIND_ROOT_PATH_MODE_LIBRARY ONLY)
set(CMAKE_FIND_ROOT_PATH_MODE_INCLUDE ONLY)
set(CMAKE_FIND_ROOT_PATH_MODE_PACKAGE ONLY)
set(HPX_WITH_PARCELPORT_TCP ON CACHE BOOL "")
set(HPX_WITH_PARCELPORT_MPI ON CACHE BOOL "")
set(HPX_PARCELPORT_LIBFABRIC_PROVIDER "gni" CACHE STRING "See libfabric docs for details, gni,verbs,psm2 etc etc")
set(HPX_PARCELPORT_LIBFABRIC_THROTTLE_SENDS "256" CACHE STRING "Max number of messages in flight at once")
set(HPX_PARCELPORT_LIBFABRIC_WITH_DEV_MODE OFF CACHE BOOL "Custom libfabric logging flag")
set(HPX_PARCELPORT_LIBFABRIC_WITH_LOGGING OFF CACHE BOOL "Libfabric parcelport logging on/off flag")
set(HPX_WITH_ZERO_COPY_SERIALIZATION_THRESHOLD "4096" CACHE STRING "The threshold in bytes to when perform zero copy optimizations (default: 128)")
# Set the TBBMALLOC_PLATFORM correctly so that find_package(TBBMalloc) sets the
# right hints
set(TBBMALLOC_PLATFORM "mic-knl" CACHE STRING "")
# We have a bunch of cores on the MIC ... increase the default
set(HPX_WITH_MAX_CPU_COUNT "512" CACHE STRING "")
# We do a cross compilation here ...
set(CMAKE_CROSSCOMPIRING ON CACHE BOOL "")
# RDTSCP is available on Xeon/Phis
set(HPX_WITH_RDTSCP ON CACHE BOOL "")

CrayKNLStatic

# Copyright (c) 2014-2017 Thomas Heller
# Copyright (c) 2017 Bryce Adelstein Lelbach
#
# SPDX-License-Identifier: BSL-1.0
# Distributed under the Boost Software License, Version 1.0. (See accompanying
# file LICENSE_1_0.txt or copy at http://www.boost.org/LICENSE_1_0.txt)
set(HPX_WITH_STATIC_LINKING ON CACHE BOOL "")
set(HPX_WITH_STATIC_EXE_LINKING ON CACHE BOOL "")
set_property(GLOBAL PROPERTY TARGET_SUPPORTS_SHARED_LIBS FALSE)
# Set the Cray Compiler Wrapper
set(CMAKE_CXX_COMPILER CC)
set(CMAKE_C_COMPILER cc)
set(CMAKE_Fortran_COMPILER ftn)
if (CMAKE_VERSION VERSION_GREATER 3.3.9)
  set(__includes "<INCLUDES>"
endif()
set(CMAKE_C_FLAGS_INIT "")
set(CMAKE_C_COMPILE_OBJECT "<CMAKE_C_COMPILER> -static -fPIC <DEFINES> ${__includes}
  <FLAGS> -o <OBJECT> -c <SOURCE>" CACHE STRING "")
set(CMAKE_CXX_FLAGS_INIT "")
set(CMAKE_CXX_COMPILE_OBJECT "<CMAKE_CXX_COMPILER> -static -fPIC <DEFINES> ${__includes}
  <FLAGS> -o <OBJECT> -c <SOURCE>" CACHE STRING "")
set(CMAKE_C_LINK_EXECUTABLE "<CMAKE_C_COMPILER> -fPIC <FLAGS> <CMAKE_C_LINK_FLAGS>
  <LINK_FLAGS> <OBJECTS> -o <TARGET> <LINK_LIBRARIES>" CACHE STRING "")
set(CMAKE_CXX_LINK_EXECUTABLE "<CMAKE_CXX_COMPILER> -fPIC <FLAGS> <CMAKE_CXX_LINK_FLAGS>
  <LINK_FLAGS> <OBJECTS> -o <TARGET> <LINK_LIBRARIES>" CACHE STRING "")
set(CMAKE_Fortran_FLAGS_INIT "")
set(CMAKE_Fortran_COMPILE_OBJECT "<CMAKE_Fortran_COMPILER> -static -fPIC <DEFINES> ${__includes}
  <FLAGS> -o <OBJECT> -c <SOURCE>" CACHE STRING "")
set(CMAKE_Fortran_LINK_EXECUTABLE "<CMAKE_Fortran_COMPILER> -fPIC <FLAGS> <CMAKE_Fortran_LINK_FLAGS>
  <LINK_FLAGS> <OBJECTS> -o <TARGET> <LINK_LIBRARIES>")
# Disable searches in the default system paths. We are cross compiling after all
# and cmake might pick up wrong libraries that way
set(CMAKE_FIND_ROOT_PATH_MODE_PROGRAM BOTH)
set(CMAKE_FIND_ROOT_PATH_MODE_LIBRARY ONLY)
set(CMAKE_FIND_ROOT_PATH_MODE_INCLUDE ONLY)
set(CMAKE_FIND_ROOT_PATH_MODE_PACKAGE ONLY)
set(HPX_WITH_PARCELPORT_TCP ON CACHE BOOL "")
set(HPX_WITH_PARCELPORT_MPI ON CACHE BOOL "")
set(HPX_WITH_PARCELPORT_MPI_MULTITHREADED ON CACHE BOOL "")
set(HPX_WITH_PARCELPORT_LIBFABRIC ON CACHE BOOL "")
set(HPX_PARCELPORT_LIBFABRIC_PROVIDER "gni" CACHE STRING
  "See libfabric docs for details, gni,verbs,psm2 etc etc")
set(HPX_PARCELPORT_LIBFABRIC_THROTTLE_SENDS "256" CACHE STRING
  "Max number of messages in flight at once")

(continues on next page)
set(HPX_PARCELPORT_LIBFABRIC_WITH_DEV_MODE OFF CACHE BOOL "Custom libfabric logging flag")
set(HPX_PARCELPORT_LIBFABRIC_WITH_LOGGING OFF CACHE BOOL "Libfabric parcelport logging on/off flag")
set(HPX_WITH_ZERO_COPY_SERIALIZE_THRESHOLD "4096" CACHE STRING "The threshold in bytes to when perform zero copy optimizations (default: 128)")
# Set the TBBMALLOC_PLATFORM correctly so that find_package(TBBMalloc) sets the
# right hints
set(TBBMALLOC_PLATFORM "mic-knl" CACHE STRING "")
# We have a bunch of cores on the MIC ... increase the default
set(HPX_WITH_MAX_CPU_COUNT "512" CACHE STRING "")
# We do a cross compilation here ...
set(CMAKE_CROSSCOMPILED ON CACHE BOOL "")
# RDTSCP is available on Xeon/Phis
set(HPX_WITH_RDTSCP ON CACHE BOOL "")

CrayStatic

# Copyright (c) 2014-2017 Thomas Heller
# Copyright (c) 2017 Bryce Adelstein Lelbach
#
# SPDX-License-Identifier: BSL-1.0
# Distributed under the Boost Software License, Version 1.0. (See accompanying
# file LICENSE_1_0.txt or copy at http://www.boost.org/LICENSE_1_0.txt)
set(HPX_PARCELPORT_LIBFABRIC_WITH_STATIC_LINKING ON CACHE BOOL "")
set(HPX_PARCELPORT_LIBFABRIC_WITH_STATIC_EXE_LINKING ON CACHE BOOL "")
set_property(GLOBAL PROPERTY TARGET_SUPPORTS_SHARED_LIBS FALSE)
# Set the Cray Compiler Wrapper
set(CMAKE_CXX_COMPILER CC)
set(CMAKE_C_COMPILER cc)
set(CMAKE_Fortran_COMPILER ftn)
if (CMAKE_VERSION VERSION_GREATER 3.3.9)
set(__includes "<INCLUDES>")
endif()
set(CMAKE_C_FLAGS_INIT "")
set(CMAKE_C_COMPILE_OBJECT "<CMAKE_C_COMPILER> -static -fPIC <DEFINES> ${__includes} <FLAGS> -o <OBJECT> -c <SOURCE>" CACHE STRING "")
set(CMAKE_C_LINK_EXECUTABLE "<CMAKE_C_COMPILER> -fPIC <FLAGS> <CMAKE_C_LINK_FLAGS> <LINK_FLAGS> <OBJECTS> -o <TARGET> <LINK_LIBRARIES>" CACHE STRING "")
set(CMAKE_CXX_FLAGS_INIT "")
set(CMAKE_CXX_COMPILE_OBJECT "<CMAKE_CXX_COMPILER> -static -fPIC <DEFINES> ${__includes} <FLAGS> -o <OBJECT> -c <SOURCE>" CACHE STRING "")
set(CMAKE_CXX_LINK_EXECUTABLE "<CMAKE_CXX_COMPILER> -fPIC <FLAGS> <CMAKE_CXX_LINK_FLAGS> <LINK_FLAGS> <OBJECTS> -o <TARGET> <LINK_LIBRARIES>" CACHE STRING "")
set(CMAKE_Fortran_COMPILER ftn)
set(CMAKE_Fortran_FLAGS_INIT "")
set(CMAKE_Fortran_COMPILE_OBJECT "<CMAKE_Fortran_COMPILER> -static -fPIC <DEFINES> ${__includes} <FLAGS> -o <OBJECT> -c <SOURCE>" CACHE STRING "")
set(CMAKE_Fortran_LINK_EXECUTABLE "<CMAKE_Fortran_COMPILER> -fPIC <FLAGS> <CMAKE_Fortran_LINK_FLAGS> <LINK_FLAGS> <OBJECTS> -o <TARGET> <LINK_LIBRARIES>" CACHE STRING "")
# Disable searches in the default system paths. We are cross compiling after all
# and cmake might pick up wrong libraries that way
set(CMAKE_FIND_ROOT_PATH_MODE_PROGRAM BOTH)
set(CMAKE_FIND_ROOT_PATH_MODE_LIBRARY ONLY)
set(CMAKE_FIND_ROOT_PATH_MODE_INCLUDE ONLY)
set(CMAKE_FIND_ROOT_PATH_MODE_PACKAGE ONLY)
# We do a cross compilation here ...

set(CMAKE_CROSSCOMPILING ON CACHE BOOL "")
# RDTSCP is available on Xeon/Phis
set(HPX_WITH_RDTSCP ON CACHE BOOL "")
set(HPX_WITH_PARCELPORT_TCP ON CACHE BOOL "")
set(HPX_WITH_PARCELPORT_MPI ON CACHE BOOL "")
set(HPX_WITH_PARCELPORT_MPI_MULTITHREADED ON CACHE BOOL "")
set(HPX_WITH_PARCELPORT_LIBFABRIC ON CACHE BOOL "")
set(HPX_PARCELPORT_LIBFABRIC_PROVIDER "gni" CACHE STRING "See libfabric docs for details, gni,verbs,psm2 etc etc")
set(HPX_PARCELPORT_LIBFABRIC_THROTTLE_SENDS "256" CACHE STRING "Max number of messages in flight at once")
set(HPX_PARCELPORT_LIBFABRIC_WITH_DEV_MODE OFF CACHE BOOL "Custom libfabric logging flag")
set(HPX_PARCELPORT_LIBFABRIC_WITH_LOGGING OFF CACHE BOOL "Libfabric parcelport logging on/off flag")
set(HPX_WITH_ZERO_COPY_SERIALIZATION_THRESHOLD "4096" CACHE STRING "The threshhold in bytes to when perform zero copy optimizations (default: 128")

XeonPhi

# Copyright (c) 2014 Thomas Heller
# SPDX-License-Identifier: BSL-1.0
# Distributed under the Boost Software License, Version 1.0. (See accompanying
# file LICENSE_1_0.txt or copy at http://www.boost.org/LICENSE_1_0.txt)
# This is the default toolchain file to be used with Intel Xeon PHIs. It sets
# the appropriate compile flags and compiler such that HPX will compile.
# Note that you still need to provide Boost, hwloc and other utility libraries
# like a custom allocator yourself.
# set(CMAKE_SYSTEM_NAME Linux)
# Set the Intel Compiler
set(CMAKE_CXX_COMPILER icpc)
set(CMAKE_C_COMPILER icc)
set(CMAKE_Fortran_COMPILER ifort)
# Add the -mmic compile flag such that everything will be compiled for the correct
# platform
set(CMAKE_CXX_FLAGS_INIT "-mmic" CACHE STRING "Initial compiler flags used to compile
--for the Xeon Phi")
set(CMAKE_C_FLAGS_INIT "-mmic" CACHE STRING "Initial compiler flags used to compile
--for the Xeon Phi")
set(CMAKE_Fortran_FLAGS_INIT "-mmic" CACHE STRING "Initial compiler flags used to
--compile for the Xeon Phi")
# Disable searches in the default system paths. We are cross compiling after all
# and cmake might pick up wrong libraries that way
set(CMAKE_FIND_ROOT_PATH_MODE_PROGRAM BOTH)
set(CMAKE_FIND_ROOT_PATH_MODE_LIBRARY ONLY)
set(CMAKE_FIND_ROOT_PATH_MODE_INCLUDE ONLY)
set(CMAKE_FIND_ROOT_PATH_MODE_PACKAGE ONLY)
# We do a cross compilation here ...
set(CMAKE_CROSSCOMPILING ON)
# Set our platform name
set(HPX_PLATFORM "XeonPhi")
# Always disable the ibverbs parcelport as it is non-functional on the BGQ.
set(HPX_WITH_PARCELPORT_VERBS OFF CACHE BOOL "Enable the ibverbs based parcelport...
--This is currently an experimental feature")

Chapter 2. What’s so special about HPX?
CMake variables used to configure HPX

In order to configure HPX, you can set a variety of options to allow CMake to generate your specific makefiles/project files.

Variables that influence how HPX is built

The options are split into these categories:

- **Generic options**
- **Build Targets options**
- **Thread Manager options**
- **AGAS options**
- **Parcelport options**
- **Profiling options**
- **Debugging options**
- **Modules options**

**Generic options**

- `HPX_WITH_ACTION_BASE_COMPATIBILITY:BOOL`
- `HPX_WITH_AUTOMATIC.Serialization_REGISTRATION:BOOL`
- `HPX_WITH_BENCHMARK_SCRIPTS_PATH:PATH`
- `HPX_WITH_BUILD_BINARY_PACKAGE:BOOL`
- `HPX_WITH_COMPILER_WARNINGS:BOOL`
- `HPX_WITH_COMPILER_WARNINGS_AS_ERRORS:BOOL`
- `HPX_WITH_COMPRESSION_BZIP2:BOOL`
- `HPX_WITH_COMPRESSION_SNAPPY:BOOL`
- `HPX_WITH_COMPRESSION_ZLIB:BOOL`
- `HPX_WITH_CUDA:BOOL`
- `HPX_WITH_CUDA_CLANG:BOOL`
• HPX_WITH_CXX14_RETURN_TYPE_DEDUCTION:BOOL
• HPX_WITH_DATAPAR_VC:BOOL
• HPX_WITH_DEPRECATION_WARNINGS:BOOL
• HPX_WITH_DISABLED_SIGNAL_EXCEPTION_HANDLERS:BOOL
• HPX_WITH_DYNAMIC_HPX_MAIN:BOOL
• HPX_WITH_FAULT_TOLERANCE:BOOL
• HPX_WITH_FORTRAN:BOOL
• HPX_WITH_FULL_RPATH:BOOL
• HPX_WITH_GCC_VERSION_CHECK:BOOL
• HPX_WITH_GENERIC_CONTEXT_COROUTINES:BOOL
• HPX_WITH_HCC:BOOL
• HPX_WITH_HIDDEN_VISIBILITY:BOOL
• HPX_WITH_LOGGING:BOOL
• HPX_WITH_MALLOC:STRING
• HPX_WITH_NATIVE_TLS:BOOL
• HPX_WITH_NICE_THREADLEVEL:BOOL
• HPX_WITH_PARCEL_COALESCLING:BOOL
• HPX_WITH_REGISTER_THREAD_COMPATIBILITY:BOOL
• HPX_WITH_RUN_MAIN_EVERYWHERE:BOOL
• HPX_WITH_STACKOVERFLOW_DETECTION:BOOL
• HPX_WITH_STATIC_LINKING:BOOL
• HPX_WITH_SYCL:BOOL
• HPX_WITH_VIM_YCM:BOOL
• HPX_WITH_ZERO_COPY_SERIALIZATION_THRESHOLD:STRING

HPX_WITH_ACTION_BASE_COMPATIBILITY:BOOL
Enable deprecated action bases (default: ON)

HPX_WITH_AUTOMATIC.Serialization.REGISTRATION:BOOL
Use automatic serialization registration for actions and functions. This affects compatibility between HPX applications compiled with different compilers (default ON)

HPX_WITH_BENCHMARK_SCRIPTS.Path:PATH
Directory to place batch scripts in

HPX_WITH_BUILD_BINARY_PACKAGE:BOOL
Build HPX on the build infrastructure on any LINUX distribution (default: OFF).

HPX_WITH_COMPILER_WARNINGS:BOOL
Enable compiler warnings (default: ON)

HPX_WITH_COMPILER_WARNINGS_AS_ERRORS:BOOL
Turn compiler warnings into errors (default: OFF)

HPX_WITH_COMPRESSION_BZIP2:BOOL
Enable bzip2 compression for parcel data (default: OFF).
HPX_WITH_COMPRESSION_SNAPPY:BOOL
   Enable snappy compression for parcel data (default: OFF).

HPX_WITH_COMPRESSION_ZLIB:BOOL
   Enable zlib compression for parcel data (default: OFF).

HPX_WITH_CUDA:BOOL
   Enable CUDA support (default: OFF)

HPX_WITH_CUDA_CLANG:BOOL
   Use clang to compile CUDA code (default: OFF)

HPX_WITH_CXX14_RETURN_TYPE_DEDUCTION:BOOL
   Enable the use of auto as a return value in some places. Overriding this flag is only necessary if the C++ compiler
   is not standard compliant, e.g. nvcc.

HPX_WITH_DATAPAR_VC:BOOL
   Enable data parallel algorithm support using the external Vc library (default: OFF)

HPX_WITH_DEPRECATED_WARNINGS:BOOL
   Enable warnings for deprecated facilities. (default: ON)

HPX_WITH_DISABLED_SIGNAL_EXCEPTION_HANDLERS:BOOL
   Disables the mechanism that produces debug output for caught signals and unhandled exceptions (default: OFF)

HPX_WITH_DYNAMIC_HPX_MAIN:BOOL
   Enable dynamic overload of system main() (Linux only, default: ON)

HPX_WITH_FAULT_TOLERANCE:BOOL
   Build HPX to tolerate failures of nodes, i.e. ignore errors in active communication channels (default: OFF)

HPX_WITH_FORTRAN:BOOL
   Enable or disable the compilation of Fortran examples using HPX

HPX_WITH_FULL_RPATH:BOOL
   Build and link HPX libraries and executables with full RPATHs (default: ON)

HPX_WITH_GCC_VERSION_CHECK:BOOL
   Don’t ignore version reported by gcc (default: ON)

HPX_WITH_GENERIC_CONTEXT_COROUTINES:BOOL
   Use Boost.Context as the underlying coroutines context switch implementation.

HPX_WITH_HCC:BOOL
   Enable hcc support (default: OFF)

HPX_WITH_HIDDEN_VISIBILITY:BOOL
   Use -fvisibility=hidden for builds on platforms which support it (default ON)

HPX_WITH_LOGGING:BOOL
   Build HPX with logging enabled (default: ON).

HPX_WITH_MALLOC:STRING
   Define which allocator should be linked in. Options are: system, tcmalloc, jemalloc, mimalloc, tbbmalloc, and
   custom (default is: tcmalloc)

HPX_WITH_NATIVE_TLS:BOOL
   Use native TLS support if available (default: ON)

HPX_WITH_NICE_THREADLEVEL:BOOL
   Set HPX worker threads to have high NICE level (may impact performance) (default: OFF)

HPX_WITH_PARCEL_COALESCING:BOOL
   Enable the parcel coalescing plugin (default: ON).
HPX WITH REGISTER THREAD COMPATIBILITY: BOOL
Enable deprecated register_thread/work functions in the hpx::applier namespace (default: ON)

HPX WITH RUN MAIN EVERYWHERE: BOOL
Run hpx_main by default on all localities (default: OFF).

HPX WITH STACKOVERFLOW DETECTION: BOOL
Enable stackoverflow detection for HPX threads/coroutines. (default: OFF, debug: ON)

HPX WITH STATIC LINKING: BOOL
Compile HPX statically linked libraries (Default: OFF)

HPX WITH SYCL: BOOL
Enable sycl support (default: OFF)

HPX WITH VIM YCM: BOOL
Generate HPX completion file for VIM YouCompleteMe plugin

HPX WITH ZERO COPY SERIALIZATION THRESHOLD: STRING
The threshold in bytes to when perform zero copy optimizations (default: 128)

Build Targets options

• HPX WITH_COMPILE_ONLY_TESTS: BOOL
• HPX WITH_DEFAULT_TARGETS: BOOL
• HPX WITH DOCUMENTATION: BOOL
• HPX WITH DOCUMENTATION_OUTPUT_FORMATS: STRING
• HPX WITH EXAMPLES: BOOL
• HPX WITH EXAMPLES_HDF5: BOOL
• HPX WITH EXAMPLES_OPENMP: BOOL
• HPX WITH EXAMPLES_QT4: BOOL
• HPX WITH EXAMPLES_QTHREADS: BOOL
• HPX WITH EXAMPLES_TBB: BOOL
• HPX WITH EXECUTABLE_PREFIX: STRING
• HPX WITH FAIL_COMPILE_TESTS: BOOL
• HPX WITH IO_COUNTERS: BOOL
• HPX WITH PSEUDO_DEPENDENCIES: BOOL
• HPX WITH TESTS: BOOL
• HPX WITH TESTS_BENCHMARKS: BOOL
• HPX WITH TESTS_EXAMPLES: BOOL
• HPX WITH TESTS_EXTERNAL_BUILD: BOOL
• HPX WITH TESTS_HEADERS: BOOL
• HPX WITH TESTS_REGRESSIONS: BOOL
• HPX WITH TESTS_UNIT: BOOL
• HPX WITH TOOLS: BOOL
HPX_WITH_COMPILE_ONLY_TESTS:BOOL
Create build system support for compile time only HPX tests (default ON)

HPX_WITH_DEFAULT_TARGETS:BOOL
Associate the core HPX library with the default build target (default: ON).

HPX_WITH_DOCUMENTATION:BOOL
Build the HPX documentation (default OFF).

HPX_WITH_DOCUMENTATION_OUTPUT_FORMATS:STRING
List of documentation output formats to generate. Valid options are html;singlehtml;latexpdf;man. Multiple values can be separated with semicolons. (default html).

HPX_WITH_EXAMPLES:BOOL
Build the HPX examples (default ON)

HPX_WITH_EXAMPLES_HDF5:BOOL
Enable examples requiring HDF5 support (default: OFF).

HPX_WITH_EXAMPLES_OPENMP:BOOL
Enable examples requiring OpenMP support (default: OFF).

HPX_WITH_EXAMPLES_QT4:BOOL
Enable examples requiring Qt4 support (default: OFF).

HPX_WITH_EXAMPLES_QTHREADS:BOOL
Enable examples requiring QThreads support (default: OFF).

HPX_WITH_EXAMPLES_TBB:BOOL
Enable examples requiring TBB support (default: OFF).

HPX_WITH_EXECUTABLE_PREFIX:STRING
Executable prefix (default none), ‘hpx_’ useful for system install.

HPX_WITH_FAIL_COMPILE_TESTS:BOOL
Create build system support for fail compile HPX tests (default ON)

HPX_WITH_IO_COUNTERS:BOOL
Enable IO counters (default: ON)

HPX_WITH_PSEUDO_DEPENDENCIES:BOOL
Force creating pseudo targets and pseudo dependencies (default ON).

HPX_WITH_TESTS:BOOL
Build the HPX tests (default ON)

HPX_WITH_TESTS_BENCHMARKS:BOOL
Build HPX benchmark tests (default: ON)

HPX_WITH_TESTS_EXAMPLES:BOOL
Add HPX examples as tests (default: ON)

HPX_WITH_TESTS_EXTERNAL_BUILD:BOOL
Build external cmake build tests (default: ON)

HPX_WITH_TESTS_HEADERS:BOOL
Build HPX header tests (default: OFF)

HPX_WITH_TESTS_REGRESSIONS:BOOL
Build HPX regression tests (default: ON)

HPX_WITH_TESTS_UNIT:BOOL
Build HPX unit tests (default: ON)
HPX Documentation,  1.4.1

**HPX_WITH_TOOLS**: BOOL

Build HPX tools (default: OFF)

**Thread Manager options**

- **HPX_SCHEDULER_MAX_TERMINATED_THREADS**: STRING
- **HPX_WITH_COROUTINE_COUNTERS**: BOOL
- **HPX_WITH_IO_POOL**: BOOL
- **HPX_WITH_MAX_CPU_COUNT**: STRING
- **HPX_WITH_MAX_NUMA_DOMAIN_COUNT**: STRING
- **HPX_WITH_MORE_THAN_64_THREADS**: BOOL
- **HPX_WITH_SCHEDULER_LOCAL_STORAGE**: BOOL
- **HPX_WITH_SPINLOCK_DEADLOCK_DETECTION**: BOOL
- **HPX_WITH_SPINLOCK_POOL_NUM**: STRING
- **HPX_WITH_STACKTRACES**: BOOL
- **HPX_WITH_SWAP_CONTEXT_EMULATION**: BOOL
- **HPX_WITH_THREAD_BACKTRACE_DEPTH**: STRING
- **HPX_WITH_THREAD_BACKTRACE_ON_SUSPENSION**: BOOL
- **HPX_WITH_THREAD_CREATION_AND_CLEANUP_RATES**: BOOL
- **HPX_WITH_THREAD_CUMULATIVE_COUNTS**: BOOL
- **HPX_WITH_THREAD_IDLE_RATES**: BOOL
- **HPX_WITH_THREAD_LOCAL_STORAGE**: BOOL
- **HPX_WITH_THREAD_MANAGER_IDLE_BACKOFF**: BOOL
- **HPX_WITH_THREAD_QUEUE_WAITTIME**: BOOL
- **HPX_WITH_THREAD_SCHEDULERS**: STRING
- **HPX_WITH_THREAD_STACK_MMAP**: BOOL
- **HPX_WITH_THREAD_STEALING_COUNTS**: BOOL
- **HPX_WITH_THREAD_TARGET_ADDRESS**: BOOL
- **HPX_WITH_TIMER_POOL**: BOOL

**HPX_SCHEDULER_MAX_TERMINATED_THREADS**: STRING

[Deprecated] Maximum number of terminated threads collected before those are cleaned up (default: 100)

**HPX_WITH_COROUTINE_COUNTERS**: BOOL

Enable keeping track of coroutine creation and rebind counts (default: OFF)

**HPX_WITH_IO_POOL**: BOOL

Disable internal IO thread pool, do not change if not absolutely necessary (default: ON)

**HPX_WITH_MAX_CPU_COUNT**: STRING

HPX applications will not use more that this number of OS-Threads (empty string means dynamic) (default: 64)

**HPX_WITH_MAX_NUMA_DOMAIN_COUNT**: STRING

HPX applications will not run on machines with more NUMA domains (default: 8)
HPX applications will be able to run on more than 64 cores (This variable is deprecated. The value is derived from HPX\_WITH\_MAX\_CPU\_COUNT instead.)

HPX\_WITH\_SCHEDULER\_LOCAL\_STORAGE\_BOOLEAN
Enable scheduler local storage for all HPX schedulers (default: OFF)

HPX\_WITH\_SPINLOCK\_DEADLOCK\_DETECTION\_BOOLEAN
Enable spinlock deadlock detection (default: OFF)

HPX\_WITH\_SPINLOCK\_POOL\_NUM\_STRING
Number of elements a spinlock pool manages (default: 128)

HPX\_WITH\_STACKTRACES\_BOOLEAN
Attach backtraces to HPX exceptions (default: ON)

HPX\_WITH\_SWAP\_CONTEXT\_EMULATION\_BOOLEAN
Emulate SwapContext API for coroutines (default: OFF)

HPX\_WITH\_THREAD\_BACKTRACE\_DEPTH\_STRING
Thread stack back trace depth being captured (default: 5)

HPX\_WITH\_THREAD\_BACKTRACE\_ON\_SUSPENSION\_BOOLEAN
Enable thread stack back trace being captured on suspension (default: OFF)

HPX\_WITH\_THREAD\_CREATION\_AND\_CLEANUP\_RATES\_BOOLEAN
Enable measuring thread creation and cleanup times (default: OFF)

HPX\_WITH\_THREAD\_CUMULATIVE\_COUNTS\_BOOLEAN
Enable keeping track of cumulative thread counts in the schedulers (default: ON)

HPX\_WITH\_THREAD\_IDLE\_RATES\_BOOLEAN
Enable measuring the percentage of overhead times spent in the scheduler (default: OFF)

HPX\_WITH\_THREAD\_LOCAL\_STORAGE\_BOOLEAN
Enable thread local storage for all HPX threads (default: OFF)

HPX\_WITH\_THREAD\_MANAGER\_IDLE\_BACKOFF\_BOOLEAN
HPX scheduler threads do exponential backoff on idle queues (default: ON)

HPX\_WITH\_THREAD\_QUEUE\_WAITTIME\_BOOLEAN
Enable collecting queue wait times for threads (default: OFF)

HPX\_WITH\_THREAD\_SCHEDULERS\_STRING
Which thread schedulers are built. Options are: all, abp-priority, local, static-priority, static, shared-priority. For multiple enabled schedulers, separate with a semicolon (default: all)

HPX\_WITH\_THREAD\_STACK\_MMAP\_BOOLEAN
Use mmap for stack allocation on appropriate platforms

HPX\_WITH\_THREAD\_STEALING\_COUNTS\_BOOLEAN
Enable keeping track of counts of thread stealing incidents in the schedulers (default: OFF)

HPX\_WITH\_THREAD\_TARGET\_ADDRESS\_BOOLEAN
Enable storing target address in thread for NUMA awareness (default: OFF)

HPX\_WITH\_TIMER\_POOL\_BOOLEAN
Disable internal timer thread pool, do not change if not absolutely necessary (default: ON)

AGAS options

- HPX\_WITH\_AGAS\_DUMP\_REFCNT\_ENTRIES\_BOOLEAN
HPX_WiTH_AGAS_DUMP_REFCNT_ENTRIES:BOOL
   Enable dumps of the AGAS refcnt tables to logs (default: OFF)

Parcelport options

• HPX_WiTH_NETWiRING:BOOL
• HPX_WiTH_PARCELPORTP_ACTION_COUNTERS:BOOL
• HPX_WiTH_PARCELPORTP_LIBFABRiC:BOOL
• HPX_WiTH_PARCELPORTP_MPI:BOOL
• HPX_WiTH_PARCELPORTP_MPI_ENV:STRING
• HPX_WiTH_PARCELPORTP_MPI_MULTITHREADED:BOOL
• HPX_WiTH_PARCELPORTP_TCP:BOOL
• HPX_WiTH_PARCELPORTP_VERBS:BOOL
• HPX_WiTH_PARCEL_PROFiLiNG:BOOL

HPX_WiTH_NETWiRING:BOOL
   Enable support for networking and multi-node runs (default: ON)

HPX_WiTH_PARCELPORTP_ACTION_COUNTERS:BOOL
   Enable performance counters reporting parcelport statistics on a per-action basis.

HPX_WiTH_PARCELPORTP_LIBFABRiC:BOOL
   Enable the libfabric based parcelport. This is currently an experimental feature

HPX_WiTH_PARCELPORTP_MPI:BOOL
   Enable the MPI based parcelport.

HPX_WiTH_PARCELPORTP_MPI_ENV:STRING
   List of environment variables checked to detect MPI (default: MV2_COMM_WORLD_RANK;PMI_RANK;OMPI_COMM_WORLD_SIZE;ALPS_APP_PE;PMIX_RANK).

HPX_WiTH_PARCELPORTP_MPI_MULTITHREADED:BOOL
   Turn on MPI multithreading support (default: ON).

HPX_WiTH_PARCELPORTP_TCP:BOOL
   Enable the TCP based parcelport.

HPX_WiTH_PARCELPORTP_VERBS:BOOL
   Enable the ibverbs based parcelport. This is currently an experimental feature

HPX_WiTH_PARCEL_PROFiLiNG:BOOL
   Enable profiling data for parcels

Profiling options

• HPX_WiTH_APEX:BOOL
• HPX_WiTH_GOOGLE_PERFTOOLs:BOOL
• HPX_WiTH_ITTNOTFiLy:BOOL
• HPX_WiTH_PAPI:BOOL

HPX_WiTH_APEX:BOOL
   Enable APEX instrumentation support.

HPX_WiTH_GOOGLE_PERFTOOLs:BOOL
   Enable Google Perftools instrumentation support.
HPX_WITH_ITTNOTIFY:BOOL
   Enable Amplifier (ITT) instrumentation support.

HPX_WITH_PAPI:BOOL
   Enable the PAPI based performance counter.

Debugging options

- HPX_WITH_ATTACH_DEBUGGER_ON_TEST_FAILURE:BOOL
- HPX_WITH_SANITIZERS:BOOL
- HPX_WITH_TESTS_DEBUG_LOG:BOOL
- HPX_WITH_TESTS_DEBUG_LOG_DESTINATION:STRING
- HPX_WITH_THREAD_DEBUG_INFO:BOOL
- HPX_WITH_THREAD_DESCRIPTION_FULL:BOOL
- HPX_WITH_THREAD_GUARD_PAGE:BOOL
- HPX_WITH_VALGRIND:BOOL
- HPX_WITH_VERIFY_LOCKS:BOOL
- HPX_WITH_VERIFY_LOCKS_BACKTRACE:BOOL
- HPX_WITH_VERIFY_LOCKS_GLOBALLY:BOOL

HPX_WITH_ATTACH_DEBUGGER_ON_TEST_FAILURE:BOOL
   Break the debugger if a test has failed (default: OFF)

HPX_WITH_SANITIZERS:BOOL
   Configure with sanitizer instrumentation support.

HPX_WITH_TESTS_DEBUG_LOG:BOOL
   Turn on debug logs (--hpx:debug-hpx-log) for tests (default: OFF)

HPX_WITH_TESTS_DEBUG_LOG_DESTINATION:STRING
   Destination for test debug logs (default: cout)

HPX_WITH_THREAD_DEBUG_INFO:BOOL
   Enable thread debugging information (default: OFF, implicitly enabled in debug builds)

HPX_WITH_THREAD_DESCRIPTION_FULL:BOOL
   Use function address for thread description (default: OFF)

HPX_WITH_THREAD_GUARD_PAGE:BOOL
   Enable thread guard page (default: ON)

HPX_WITH_VALGRIND:BOOL
   Enable Valgrind instrumentation support.

HPX_WITH_VERIFY_LOCKS:BOOL
   Enable lock verification code (default: OFF, implicitly enabled in debug builds)

HPX_WITH_VERIFY_LOCKS_BACKTRACE:BOOL
   Enable thread stack back trace being captured on lock registration (to be used in combination with HPX_WITH_VERIFY_LOCKS=ON, default: OFF)

HPX_WITH_VERIFY_LOCKS_GLOBALLY:BOOL
   Enable global lock verification code (default: OFF, implicitly enabled in debug builds)
HPX Documentation, 1.4.1

Modules options

- HPX_AFFINITY_WITH_COMPATIBILITY_HEADERS:BOOL
- HPX_AFFINITY_WITH_DEPRECATION_WARNINGS:BOOL
- HPX_AFFINITY_WITH_TESTS:BOOL
- HPX_ALGORITHMS_WITH_DEPRECATION_WARNINGS:BOOL
- HPX_ALGORITHMS_WITH_TESTS:BOOL
- HPX_ALLOCATOR_SUPPORT_WITH_COMPATIBILITY_HEADERS:BOOL
- HPX_ALLOCATOR_SUPPORT_WITH_DEPRECATION_WARNINGS:BOOL
- HPX_ALLOCATOR_SUPPORT_WITH_TESTS:BOOL
- HPX_ASSERTION_WITH_COMPATIBILITY_HEADERS:BOOL
- HPX_ASSERTION_WITH_DEPRECATION_WARNINGS:BOOL
- HPX_ASSERTION_WITH_TESTS:BOOL
- HPX_BASIC_EXECUTION_WITH_COMPATIBILITY_HEADERS:BOOL
- HPX_BASIC_EXECUTION_WITH_DEPRECATION_WARNINGS:BOOL
- HPX_BASIC_EXECUTION_WITH_TESTS:BOOL
- HPX_BATCH_ENVIRONMENTS_WITH_COMPATIBILITY_HEADERS:BOOL
- HPX_BATCH_ENVIRONMENTS_WITH_DEPRECATION_WARNINGS:BOOL
- HPX_BATCH_ENVIRONMENTS_WITH_TESTS:BOOL
- HPX_CACHE_WITH_COMPATIBILITY_HEADERS:BOOL
- HPX_CACHE_WITH_DEPRECATION_WARNINGS:BOOL
- HPX_CACHE_WITH_TESTS:BOOL
- HPX_CHECKPOINT_WITH_COMPATIBILITY_HEADERS:BOOL
- HPX_CHECKPOINT_WITH_DEPRECATION_WARNINGS:BOOL
- HPX_CHECKPOINT_WITH_TESTS:BOOL
- HPX_COLLECTIVES_WITH_COMPATIBILITY_HEADERS:BOOL
- HPX_COLLECTIVES_WITH_DEPRECATION_WARNINGS:BOOL
- HPX_COLLECTIVES_WITH_TESTS:BOOL
- HPX_COMPUTE_CUDA_WITH_DEPRECATION_WARNINGS:BOOL
- HPX_COMPUTE_CUDA_WITH_TESTS:BOOL
- HPX_COMPUTE_WITH_COMPATIBILITY_HEADERS:BOOL
- HPX_COMPUTE_WITH_DEPRECATION_WARNINGS:BOOL
- HPX_COMPUTE_WITH_TESTS:BOOL
- HPX_CONCEPTS_WITH_COMPATIBILITY_HEADERS:BOOL
- HPX_CONCEPTS_WITH_DEPRECATION_WARNINGS:BOOL
- HPX_CONCEPTS_WITH_TESTS:BOOL
- HPX_CONCURRENCY_WITH_COMPATIBILITY_HEADERS:BOOL
- HPX_CONCURRENCY_WITH_DEPRECATION_WARNINGS:BOOL
• HPX_CONCURRENCY_WITH_TESTS:BOOL
• HPX_CONFIG_WITH_COMPATIBILITY_HEADERS:BOOL
• HPX_CONFIG_WITH_DEPRECATION_WARNINGS:BOOL
• HPX_CONFIG_WITH_TESTS:BOOL
• HPX_COROUTINES_WITH_COMPATIBILITY_HEADERS:BOOL
• HPX_COROUTINES_WITH_DEPRECATION_WARNINGS:BOOL
• HPX_COROUTINES_WITH_TESTS:BOOL
• HPX_DATASTRUCTURES_WITH_ADAPT_STD_TUPLE:BOOL
• HPX_DATASTRUCTURES_WITH_COMPATIBILITY_HEADERS:BOOL
• HPX_DATASTRUCTURES_WITH_DEPRECATION_WARNINGS:BOOL
• HPX_DATASTRUCTURES_WITH_TESTS:BOOL
• HPX_DEBUGGING_WITH_COMPATIBILITY_HEADERS:BOOL
• HPX_DEBUGGING_WITH_DEPRECATION_WARNINGS:BOOL
• HPX_DEBUGGING_WITH_TESTS:BOOL
• HPX_ERRORS_WITH_COMPATIBILITY_HEADERS:BOOL
• HPX_ERRORS_WITH_DEPRECATION_WARNINGS:BOOL
• HPX_ERRORS_WITH_TESTS:BOOL
• HPX_EXECUTION_WITH_DEPRECATION_WARNINGS:BOOL
• HPX_EXECUTION_WITH_TESTS:BOOL
• HPX_FILESYSTEM_WITH_BOOST_FILESYSTEM_COMPATIBILITY:BOOL
• HPX_FILESYSTEM_WITH_DEPRECATION_WARNINGS:BOOL
• HPX_FILESYSTEM_WITH_TESTS:BOOL
• HPX_FORMAT_WITH_COMPATIBILITY_HEADERS:BOOL
• HPX_FORMAT_WITH_DEPRECATION_WARNINGS:BOOL
• HPX_FORMAT_WITH_TESTS:BOOL
• HPX_FUNCTIONAL_WITH_COMPATIBILITY_HEADERS:BOOL
• HPX_FUNCTIONAL_WITH_DEPRECATION_WARNINGS:BOOL
• HPX_FUNCTIONAL_WITH_TESTS:BOOL
• HPX_HARDWARE_WITH_COMPATIBILITY_HEADERS:BOOL
• HPX_HARDWARE_WITH_DEPRECATION_WARNINGS:BOOL
• HPX_HARDWARE_WITH_TESTS:BOOL
• HPX_HASHING_WITH_COMPATIBILITY_HEADERS:BOOL
• HPX_HASHING_WITH_DEPRECATION_WARNINGS:BOOL
• HPX_HASHING_WITH_TESTS:BOOL
• HPX_ITERATOR_SUPPORT_WITH_COMPATIBILITY_HEADERS:BOOL
• HPX_ITERATOR_SUPPORT_WITH_DEPRECATION_WARNINGS:BOOL
• `HPX_ITERATOR_SUPPORT_WITH_TESTS:BOOL`
• `HPX_LOCAL_LCOS_WITH_COMPATIBILITY_HEADERS:BOOL`
• `HPX_LOCAL_LCOS_WITH_DEPRECATION_WARNINGS:BOOL`
• `HPX_LOCAL_LCOS_WITH_TESTS:BOOL`
• `HPX_LOGGING_WITH_COMPATIBILITY_HEADERS:BOOL`
• `HPX_LOGGING_WITH_DEPRECATION_WARNINGS:BOOL`
• `HPX_LOGGING_WITH_TESTS:BOOL`
• `HPX_MEMORY_WITH_DEPRECATION_WARNINGS:BOOL`
• `HPX_MEMORY_WITH_TESTS:BOOL`
• `HPX_PLUGIN_WITH_COMPATIBILITY_HEADERS:BOOL`
• `HPX_PLUGIN_WITH_DEPRECATION_WARNINGS:BOOL`
• `HPX_PLUGIN_WITH_TESTS:BOOL`
• `HPX_PREPROCESSOR_WITH_COMPATIBILITY_HEADERS:BOOL`
• `HPX_PREPROCESSOR_WITH_DEPRECATION_WARNINGS:BOOL`
• `HPX_PREPROCESSOR_WITH_TESTS:BOOL`
• `HPX_PROGRAM_OPTIONS_WITH_BOOST_PROGRAM_OPTIONS_COMPATIBILITY:BOOL`
• `HPX_PROGRAM_OPTIONS_WITH_DEPRECATION_WARNINGS:BOOL`
• `HPX_PROGRAM_OPTIONS_WITH_TESTS:BOOL`
• `HPX_RESILIENCY_WITH_DEPRECATION_WARNINGS:BOOL`
• `HPX_RESILIENCY_WITH_TESTS:BOOL`
• `HPX_RESOURCE_PARTITIONER_WITH_COMPATIBILITY_HEADERS:BOOL`
• `HPX_RESOURCE_PARTITIONER_WITH_DEPRECATION_WARNINGS:BOOL`
• `HPX_RESOURCE_PARTITIONER_WITH_TESTS:BOOL`
• `HPX_SEGMENTED_ALGORITHMS_WITH_DEPRECATION_WARNINGS:BOOL`
• `HPX_SEGMENTED_ALGORITHMS_WITH_TESTS:BOOL`
• `HPX_SERIALIZATION_WITH_BOOST_TYPES:BOOL`
• `HPX_SERIALIZATION_WITH_COMPATIBILITY_HEADERS:BOOL`
• `HPX_SERIALIZATION_WITH_DEPRECATION_WARNINGS:BOOL`
• `HPX_SERIALIZATION_WITH_TESTS:BOOL`
• `HPX_STATIC_REINIT_WITH_COMPATIBILITY_HEADERS:BOOL`
• `HPX_STATIC_REINIT_WITH_DEPRECATION_WARNINGS:BOOL`
• `HPX_STATIC_REINIT_WITH_TESTS:BOOL`
• `HPX_STATISTICS_WITH_COMPATIBILITY_HEADERS:BOOL`
• `HPX_STATISTICS_WITH_DEPRECATION_WARNINGS:BOOL`
• `HPX_STATISTICS_WITH_TESTS:BOOL`
• `HPX_SYNCHRONIZATION_WITH_COMPATIBILITY_HEADERS:BOOL`
HPX_SYNCHRONIZATION_WITH_DEPRECATION_WARNINGS:BOOL
HPX_SYNCHRONIZATION_WITH_TESTS:BOOL
HPX_TESTING_WITH_COMPATIBILITY_HEADERS:BOOL
HPX_TESTING_WITH_DEPRECATION_WARNINGS:BOOL
HPX_TESTING_WITH_TESTS:BOOL
HPX_THREADMANAGER_WITH_COMPATIBILITY_HEADERS:BOOL
HPX_THREADMANAGER_WITH_DEPRECATION_WARNINGS:BOOL
HPX_THREADMANAGER_WITH_TESTS:BOOL
HPX_THREAD_SUPPORT_WITH_COMPATIBILITY_HEADERS:BOOL
HPX_THREAD_SUPPORT_WITH_DEPRECATION_WARNINGS:BOOL
HPX_THREAD_SUPPORT_WITH_TESTS:BOOL
HPX_TIMING_WITH_COMPATIBILITY_HEADERS:BOOL
HPX_TIMING_WITH_DEPRECATION_WARNINGS:BOOL
HPX_TIMING_WITH_TESTS:BOOL
HPX_TOPOLOGY_WITH_COMPATIBILITY_HEADERS:BOOL
HPX_TOPOLOGY_WITH_DEPRECATION_WARNINGS:BOOL
HPX_TOPOLOGY_WITH_TESTS:BOOL
HPX_TYPE_SUPPORT_WITH_COMPATIBILITY_HEADERS:BOOL
HPX_TYPE_SUPPORT_WITH_DEPRECATION_WARNINGS:BOOL
HPX_TYPE_SUPPORT_WITH_TESTS:BOOL
HPX_UTIL_WITH_COMPATIBILITY_HEADERS:BOOL
HPX_UTIL_WITH_DEPRECATION_WARNINGS:BOOL
HPX_UTIL_WITH_TESTS:BOOL
HPX_AFFINITY_WITH_COMPATIBILITY_HEADERS:BOOL
Enable compatibility headers for old headers. (default: ON)
HPX_AFFINITY_WITH_DEPRECATION_WARNINGS:BOOL
Enable warnings for deprecated facilities. (default: ON)
HPX_AFFINITY_WITH_TESTS:BOOL
Build HPX affinity module tests. (default: ON)
HPX_ALGORITHMS_WITH_DEPRECATION_WARNINGS:BOOL
Enable warnings for deprecated facilities. (default: ON)
HPX_ALGORITHMS_WITH_TESTS:BOOL
Build HPX algorithms module tests. (default: ON)
HPX_ALLOCATOR_SUPPORT_WITH_COMPATIBILITY_HEADERS:BOOL
Enable compatibility headers for old headers. (default: ON)
HPX_ALLOCATOR_SUPPORT_WITH_DEPRECATION_WARNINGS:BOOL
Enable warnings for deprecated facilities. (default: ON)
HPX_ALLOCATOR_SUPPORT_WITH_TESTS:BOOL
Build HPX allocator_support module tests. (default: ON)
HPX_ASSERTION_WITH_COMPATIBILITY_HEADERS:BOOL
   Enable compatibility headers for old headers. (default: ON)

HPX_ASSERTION_WITH_DEPRECATION_WARNINGS:BOOL
   Enable warnings for deprecated facilities. (default: ON)

HPX_ASSERTION_WITH_TESTS:BOOL
   Build HPX assertion module tests. (default: ON)

HPX_BASIC_EXECUTION_WITH_COMPATIBILITY_HEADERS:BOOL
   Enable compatibility headers for old headers. (default: ON)

HPX_BASIC_EXECUTION_WITH_DEPRECATION_WARNINGS:BOOL
   Enable warnings for deprecated facilities. (default: ON)

HPX_BASIC_EXECUTION_WITH_TESTS:BOOL
   Build HPX basic_execution module tests. (default: ON)

HPX_BATCH_ENVIRONMENTS_WITH_COMPATIBILITY_HEADERS:BOOL
   Enable compatibility headers for old headers. (default: ON)

HPX_BATCH_ENVIRONMENTS_WITH_DEPRECATION_WARNINGS:BOOL
   Enable warnings for deprecated facilities. (default: ON)

HPX_BATCH_ENVIRONMENTS_WITH_TESTS:BOOL
   Build HPX batch_environments module tests. (default: ON)

HPX_CACHE_WITH_COMPATIBILITY_HEADERS:BOOL
   Enable compatibility headers for old headers. (default: ON)

HPX_CACHE_WITH_DEPRECATION_WARNINGS:BOOL
   Enable warnings for deprecated facilities. (default: ON)

HPX_CACHE_WITH_TESTS:BOOL
   Build HPX cache module tests. (default: ON)

HPX_CHECKPOINT_WITH_COMPATIBILITY_HEADERS:BOOL
   Enable compatibility headers for old headers. (default: ON)

HPX_CHECKPOINT_WITH_DEPRECATION_WARNINGS:BOOL
   Enable warnings for deprecated facilities. (default: ON)

HPX_CHECKPOINT_WITH_TESTS:BOOL
   Build HPX checkpoint module tests. (default: ON)

HPX_COLLECTIVES_WITH_COMPATIBILITY_HEADERS:BOOL
   Enable compatibility headers for old headers. (default: ON)

HPX_COLLECTIVES_WITH_DEPRECATION_WARNINGS:BOOL
   Enable warnings for deprecated facilities. (default: ON)

HPX_COLLECTIVES_WITH_TESTS:BOOL
   Build HPX collectives module tests. (default: ON)

HPX_COMPUTE_CUDA_WITH_DEPRECATION_WARNINGS:BOOL
   Enable warnings for deprecated facilities. (default: ON)

HPX_COMPUTE_CUDA_WITH_TESTS:BOOL
   Build HPX compute_cuda module tests. (default: ON)

HPX_COMPUTE_WITH_DEPRECATION_WARNINGS:BOOL
   Enable warnings for deprecated facilities. (default: ON)
HPX_COMPUTE_WITH_TESTS:BOOL
Build HPX compute module tests. (default: ON)

HPX_CONCEPTS_WITH_COMPATIBILITY_HEADERS:BOOL
Enable compatibility headers for old headers. (default: ON)

HPX_CONCEPTS_WITH_DEPRECATED_WARNINGS:BOOL
Enable warnings for deprecated facilities. (default: ON)

HPX_CONCEPTS_WITH_TESTS:BOOL
Build HPX concepts module tests. (default: ON)

HPX_CONCURRENCY_WITH_COMPATIBILITY_HEADERS:BOOL
Enable compatibility headers for old headers. (default: ON)

HPX_CONCURRENCY_WITH_DEPRECATED_WARNINGS:BOOL
Enable warnings for deprecated facilities. (default: ON)

HPX_CONCURRENCY_WITH_TESTS:BOOL
Build HPX concurrency module tests. (default: ON)

HPX_CONFIG_WITH_COMPATIBILITY_HEADERS:BOOL
Enable compatibility headers for old headers. (default: OFF)

HPX_CONFIG_WITH_DEPRECATED_WARNINGS:BOOL
Enable warnings for deprecated facilities. (default: ON)

HPX_CONFIG_WITH_TESTS:BOOL
Build HPX config module tests. (default: ON)

HPX_COROUTINES_WITH_COMPATIBILITY_HEADERS:BOOL
Enable compatibility headers for old headers. (default: ON)

HPX_COROUTINES_WITH_DEPRECATED_WARNINGS:BOOL
Enable warnings for deprecated facilities. (default: ON)

HPX_COROUTINES_WITH_TESTS:BOOL
Build HPX coroutines module tests. (default: ON)

HPX_DATASTRUCTURES_WITH_COMPATIBILITY_HEADERS:BOOL
Enable compatibility headers for old headers. (default: ON)

HPX_DATASTRUCTURES_WITH_COMPATIBILITY_HEADERS:BOOL
Enable compatibility of hpx::util::tuple with std::tuple. (default: ON)

HPX_DATASTRUCTURES_WITH_DEPRECATED_WARNINGS:BOOL
Enable warnings for deprecated facilities. (default: ON)

HPX_DATASTRUCTURES_WITH_TESTS:BOOL
Build HPX datastructures module tests. (default: ON)

HPX_DEBUGGING_WITH_COMPATIBILITY_HEADERS:BOOL
Enable compatibility headers for old headers. (default: ON)

HPX_DEBUGGING_WITH_DEPRECATED_WARNINGS:BOOL
Enable warnings for deprecated facilities. (default: ON)

HPX_DEBUGGING_WITH_TESTS:BOOL
Build HPX debugging module tests. (default: ON)

HPX_ERRORS_WITH_COMPATIBILITY_HEADERS:BOOL
Enable compatibility headers for old headers. (default: ON)
HPX_ERRORS_WITH_DEPRECATION_WARNINGS:BOOL
Enable warnings for deprecated facilities. (default: ON)

HPX_ERRORS_WITH_TESTS:BOOL
Build HPX errors module tests. (default: ON)

HPX_EXECUTION_WITH_DEPRECATION_WARNINGS:BOOL
Enable warnings for deprecated facilities. (default: ON)

HPX_EXECUTION_WITH_TESTS:BOOL
Build HPX execution module tests. (default: ON)

HPX_FILESYSTEM_WITH_BOOST_FILESYSTEM_COMPATIBILITY:BOOL
Enable Boost.FileSystem compatibility. (default: OFF)

HPX_FILESYSTEM_WITH_DEPRECATION_WARNINGS:BOOL
Enable warnings for deprecated facilities. (default: ON)

HPX_FILESYSTEM_WITH_TESTS:BOOL
Build HPX filesystem module tests. (default: ON)

HPX_FORMAT_WITH_COMPATIBILITY_HEADERS:BOOL
Enable compatibility headers for old headers. (default: ON)

HPX_FORMAT_WITH_DEPRECATION_WARNINGS:BOOL
Enable warnings for deprecated facilities. (default: ON)

HPX_FORMAT_WITH_TESTS:BOOL
Build HPX format module tests. (default: ON)

HPX_FUNCTIONAL_WITH_COMPATIBILITY_HEADERS:BOOL
Enable compatibility headers for old headers. (default: ON)

HPX_FUNCTIONAL_WITH_DEPRECATION_WARNINGS:BOOL
Enable warnings for deprecated facilities. (default: ON)

HPX_FUNCTIONAL_WITH_TESTS:BOOL
Build HPX functional module tests. (default: ON)

HPX_HARDWARE_WITH_COMPATIBILITY_HEADERS:BOOL
Enable compatibility headers for old headers. (default: ON)

HPX_HARDWARE_WITH_DEPRECATION_WARNINGS:BOOL
Enable warnings for deprecated facilities. (default: ON)

HPX_HARDWARE_WITH_TESTS:BOOL
Build HPX hardware module tests. (default: ON)

HPX_HASHING_WITH_COMPATIBILITY_HEADERS:BOOL
Enable compatibility headers for old headers. (default: ON)

HPX_HASHING_WITH_DEPRECATION_WARNINGS:BOOL
Enable warnings for deprecated facilities. (default: ON)

HPX_HASHING_WITH_TESTS:BOOL
Build HPX hashing module tests. (default: ON)

HPX_ITERATOR_SUPPORT_WITH_COMPATIBILITY_HEADERS:BOOL
Enable compatibility headers for old headers. (default: ON)

HPX_ITERATOR_SUPPORT_WITH_DEPRECATION_WARNINGS:BOOL
Enable warnings for deprecated facilities. (default: ON)
HPX_ITERATOR_SUPPORT_WITH_TESTS:BOOL
  Build HPX iterator_support module tests. (default: ON)

HPX_LOCAL_LCOS_WITH_COMPATIBILITY_HEADERS:BOOL
  Enable compatibility headers for old headers. (default: ON)

HPX_LOCAL_LCOS_WITH_DEPRECATION_WARNINGS:BOOL
  Enable warnings for deprecated facilities. (default: ON)

HPX_LOCAL_LCOS_WITH_TESTS:BOOL
  Build HPX local_lcos module tests. (default: ON)

HPX_LOGGING_WITH_COMPATIBILITY_HEADERS:BOOL
  Enable compatibility headers for old headers. (default: ON)

HPX_LOGGING_WITH_DEPRECATION_WARNINGS:BOOL
  Enable warnings for deprecated facilities. (default: ON)

HPX_LOGGING_WITH_TESTS:BOOL
  Build HPX logging module tests. (default: ON)

HPX_MEMORY_WITH_DEPRECATION_WARNINGS:BOOL
  Enable warnings for deprecated facilities. (default: ON)

HPX_MEMORY_WITH_TESTS:BOOL
  Build HPX memory module tests. (default: ON)

HPX_PLUGIN_WITH_COMPATIBILITY_HEADERS:BOOL
  Enable compatibility headers for old headers. (default: ON)

HPX_PLUGIN_WITH_DEPRECATION_WARNINGS:BOOL
  Enable warnings for deprecated facilities. (default: ON)

HPX_PLUGIN_WITH_TESTS:BOOL
  Build HPX plugin module tests. (default: ON)

HPX_PREPROCESSOR_WITH_COMPATIBILITY_HEADERS:BOOL
  Enable compatibility headers for old headers. (default: ON)

HPX_PREPROCESSOR_WITH_DEPRECATION_WARNINGS:BOOL
  Enable warnings for deprecated facilities. (default: ON)

HPX_PREPROCESSOR_WITH_TESTS:BOOL
  Build HPX preprocessor module tests. (default: ON)

HPX_PROGRAM_OPTIONS_WITH_BOOST_PROGRAM_OPTIONS_COMPATIBILITY:BOOL
  Enable Boost.ProgramOptions compatibility. (default: ON)

HPX_PROGRAM_OPTIONS_WITH_DEPRECATION_WARNINGS:BOOL
  Enable warnings for deprecated facilities. (default: ON)

HPX_PROGRAM_OPTIONS_WITH_TESTS:BOOL
  Build HPX program_options module tests. (default: ON)

HPX_RESILIENCY_WITH_DEPRECATION_WARNINGS:BOOL
  Enable warnings for deprecated facilities. (default: ON)

HPX_RESILIENCY_WITH_TESTS:BOOL
  Build HPX resiliency module tests. (default: ON)

HPX_RESOURCE_PARTITIONER_WITH_COMPATIBILITY_HEADERS:BOOL
  Enable compatibility headers for old headers. (default: ON)
HPX_RESOURCE_PARTITIONER_WITH_DEPRECATION_WARNINGS:BOOL
Enable warnings for deprecated facilities. (default: ON)

HPX_RESOURCE_PARTITIONER_WITH_TESTS:BOOL
Build HPX resource_partitioner module tests. (default: ON)

HPX_SEGMENTED_ALGORITHMS_WITH_DEPRECATION_WARNINGS:BOOL
Enable warnings for deprecated facilities. (default: ON)

HPX_SEGMENTED_ALGORITHMS_WITH_TESTS:BOOL
Build HPX segmented_algorithms module tests. (default: ON)

HPX_SERIALIZATION_WITH_BOOST_TYPES:BOOL
Enable serialization of certain Boost types. (default: ON)

HPX_SERIALIZATION_WITH_COMPATIBILITY_HEADERS:BOOL
Enable compatibility headers for old headers. (default: ON)

HPX_SERIALIZATION_WITH_DEPRECATION_WARNINGS:BOOL
Enable warnings for deprecated facilities. (default: ON)

HPX_SERIALIZATION_WITH_TESTS:BOOL
Build HPX serialization module tests. (default: ON)

HPX_STATIC_REINIT_WITH_COMPATIBILITY_HEADERS:BOOL
Enable compatibility headers for old headers. (default: ON)

HPX_STATIC_REINIT_WITH_DEPRECATION_WARNINGS:BOOL
Enable warnings for deprecated facilities. (default: ON)

HPX_STATIC_REINIT_WITH_TESTS:BOOL
Build HPX static_reinit module tests. (default: ON)

HPX_STATISTICS_WITH_COMPATIBILITY_HEADERS:BOOL
Enable compatibility headers for old headers. (default: ON)

HPX_STATISTICS_WITH_DEPRECATION_WARNINGS:BOOL
Enable warnings for deprecated facilities. (default: ON)

HPX_STATISTICS_WITH_TESTS:BOOL
Build HPX statistics module tests. (default: ON)

HPX_SYNCHRONIZATION_WITH_COMPATIBILITY_HEADERS:BOOL
Enable compatibility headers for old headers. (default: ON)

HPX_SYNCHRONIZATION_WITH_DEPRECATION_WARNINGS:BOOL
Enable warnings for deprecated facilities. (default: ON)

HPX_SYNCHRONIZATION_WITH_TESTS:BOOL
Build HPX synchronization module tests. (default: ON)

HPX_TESTING_WITH_COMPATIBILITY_HEADERS:BOOL
Enable compatibility headers for old headers. (default: ON)

HPX_TESTING_WITH_DEPRECATION_WARNINGS:BOOL
Enable warnings for deprecated facilities. (default: ON)

HPX_TESTING_WITH_TESTS:BOOL
Build HPX testing module tests. (default: ON)

HPX_THREADMANAGER_WITH_COMPATIBILITY_HEADERS:BOOL
Enable compatibility headers for old headers. (default: ON)
HPX_THREADEXECUTOR Artículo con depresión: BOOL
Enable warnings for deprecated facilities. (default: ON)

HPX_THREADMANAGER Artículo con tests: BOOL
Build HPX threadmanager module tests. (default: ON)

HPX_THREAD_SUPPORT Artículo con compatibilidad CABERÁ: BOOL
Enable compatibility headers for old headers. (default: ON)

HPX_THREAD_SUPPORT Artículo con depresión WARNING: BOOL
Enable warnings for deprecated facilities. (default: ON)

HPX_THREAD_SUPPORT Artículo con tests: BOOL
Build HPX thread_support module tests. (default: ON)

HPX_TIMING Artículo con compatibilidad CABERÁ: BOOL
Enable compatibility headers for old headers. (default: ON)

HPX_TIMING Artículo con depresión WARNING: BOOL
Enable warnings for deprecated facilities. (default: ON)

HPX_TIMING Artículo con tests: BOOL
Build HPX timing module tests. (default: ON)

HPX_TOPOLOGY Artículo con compatibilidad CABERÁ: BOOL
Enable compatibility headers for old headers. (default: ON)

HPX_TOPOLOGY Artikel con depresión WARNING: BOOL
Enable warnings for deprecated facilities. (default: ON)

HPX_TOPOLOGY Artículo con tests: BOOL
Build HPX topology module tests. (default: ON)

HPX_TYPE_SUPPORT Artículo con compatibilidad CABERÁ: BOOL
Enable compatibility headers for old headers. (default: ON)

HPX_TYPE_SUPPORT Artículo con depresión WARNING: BOOL
Enable warnings for deprecated facilities. (default: ON)

HPX_TYPE_SUPPORT Artículo con tests: BOOL
Build HPX type_support module tests. (default: ON)

HPX_UTIL Artículo con compatibilidad CABERÁ: BOOL
Enable compatibility headers for old headers. (default: ON)

HPX_UTIL Artículo con depresión WARNING: BOOL
Enable warnings for deprecated facilities. (default: ON)

HPX_UTIL Artículo con tests: BOOL
Build HPX util module tests. (default: ON)

Additional tools and libraries used by HPX

Here is a list of additional libraries and tools that are either optionally supported by the build system or are optionally required for certain examples or tests. These libraries and tools can be detected by the HPX build system.

Each of the tools or libraries listed here will be automatically detected if they are installed in some standard location. If a tool or library is installed in a different location, you can specify its base directory by appending _ROOT to the variable name as listed below. For instance, to configure a custom directory for BOOST, specify BOOST_ROOT=/custom/boost/root.

BOOST_ROOT: PATH
Specifies where to look for the Boost installation to be used for compiling HPX. Set this if CMake is not able
to locate a suitable version of Boost. The directory specified here can be either the root of an installed Boost
distribution or the directory where you unpacked and built Boost without installing it (with staged libraries).

**HWLOC_ROOT:PATH**
Specifies where to look for the hwloc library. Set this if CMake is not able to locate a suitable version of hwloc.
Hwloc provides platform-independent support for extracting information about the used hardware architecture
(number of cores, number of NUMA domains, hyperthreading, etc.). **HPX** utilizes this information if available.

**PAPI_ROOT:PATH**
Specifies where to look for the PAPI library. The PAPI library is needed to compile a special component ex-
posing PAPI hardware events and counters as **HPX** performance counters. This is not available on the Windows
platform.

**AMPLIFIER_ROOT:PATH**
Specifies where to look for one of the tools of the Intel Parallel Studio product, either Intel Amplifier or Intel
Inspector. This should be set if the CMake variable **HPX_USE_ITT_NOTIFY** is set to ON. Enabling ITT
support in **HPX** will integrate any application with the mentioned Intel tools, which customizes the generated
information for your application and improves the generated diagnostics.

In addition, some of the examples may need the following variables:

**HDF5_ROOT:PATH**
Specifies where to look for the Hierarchical Data Format V5 (HDF5) include files and libraries.

### 2.5.3 Creating **HPX** projects

#### Using **HPX** with pkg-config

**How to build **HPX** applications with pkg-config**

After you are done installing **HPX**, you should be able to build the following program. It prints Hello World! on
the locality you run it on.

```cpp
// Copyright (c) 2007-2012 Hartmut Kaiser
//
// SPDX-License-Identifier: BSL-1.0
// Distributed under the Boost Software License, Version 1.0. (See accompanying
// file LICENSE_1_0.txt or copy at http://www.boost.org/LICENSE_1_0.txt)

// The purpose of this example is to execute a HPX-thread printing
// "Hello World!" once. That's all.

// [hello_world_1_getting_started
// Including 'hpx/hpx_main.hpp' instead of the usual 'hpx/hpx_init.hpp' enables
// to use the plain C-main below as the direct main HPX entry point.
#include <hpx/hpx_main.hpp>
#include <hpx/include/iostreams.hpp>

int main()
{
    // Say hello to the world!
    hpx::cout << "Hello World!\n" << hpx::flush;
    return 0;
}

//]
```

Copy the text of this program into a file called hello_world.cpp.
Now, in the directory where you put hello_world.cpp, issue the following commands (where \$HPX\_LOCATION is the build directory or CMAKE\_INSTALL\_PREFIX you used while building HPX):

```bash
export PKG_CONFIG_PATH=$PKG_CONFIG_PATH:$HPX\_LOCATION/lib/pkgconfig
c++ -o hello_world hello_world.cpp \\n   pkg-config --cflags --libs hpx_application\\
   -lhpx_iostreams -DHPX\_APPLICATION\_NAME=hello_world
```

**Important:** When using pkg-config with HPX, the pkg-config flags must go after the -o flag.

**Note:** HPX libraries have different names in debug and release mode. If you want to link against a debug HPX library, you need to use the \_debug suffix for the pkg-config name. That means instead of hpx\_application or hpx\_component, you will have to use hpx\_application\_debug or hpx\_component\_debug. Moreover, all referenced HPX components need to have an appended d suffix. For example, instead of -lhpx\_iostreams you will need to specify -lhpx\_iostreamsd.

**Important:** If the HPX libraries are in a path that is not found by the dynamic linker, you will need to add the path $HPX\_LOCATION/lib to your linker search path (for example LD\_LIBRARY\_PATH on Linux).

To test the program, type:

`.\hello_world`

which should print Hello World! and exit.

**How to build HPX components with pkg-config**

Let’s try a more complex example involving an HPX component. An HPX component is a class that exposes HPX actions. HPX components are compiled into dynamically loaded modules called component libraries. Here’s the source code:

```cpp
//hello_world_component.hpp
#include "hello_world_component.hpp"
#include <hpx/include/iostreams.hpp>

namespace examples { namespace server {
    void hello_world::invoke()
    {
        hpx::cout << "Hello HPX World!" << std::endl;
    }
}}

HPX_REGISTER_COMPONENT_MODULE();

typedef hpx::components::component<
    examples::server::hello_world > hello_world_type;
```

(continues on next page)
HPX_REGISTER_COMPONENT(hello_world_type, hello_world);
HPX_REGISTER_ACTION(}

hello_world_component.hpp

//{hello_world_hpp_getting_started
#if !defined(HELLO_WORLD_COMPONENT_HPP)
#define HELLO_WORLD_COMPONENT_HPP

#include <hpx/hpx.hpp>
#include <hpx/include/actions.hpp>
#include <hpx/include/lcos.hpp>
#include <hpx/include/components.hpp>
#include <hpx/include/serialization.hpp>
#include <utility>

namespace examples { namespace server {

struct HPX_COMPONENT_EXPORT hello_world
  : hpx::components::component_base<hello_world>
{
  void invoke();
  HPX_DEFINE_COMPONENT_ACTION(hello_world, invoke);
};
}}

HPX_REGISTER_ACTION_DECLARATION(
examples::server::hello_world::invoke_action, hello_world_invoke_action);

namespace examples { 

struct hello_world
  : hpx::components::client_base<hello_world, server::hello_world>
{
  typedef hpx::components::client_base<hello_world, server::hello_world>
    base_type;

  hello_world(hpx::future<hpx::naming::id_type> && f)
    : base_type(std::move(f))
  {{
  }

  hello_world(hpx::naming::id_type && f)
    : base_type(std::move(f))
  {{
  }

  void invoke()
  {
    hpx::async<server::hello_world::invoke_action>(this->get_id()).get();
  }
};
}

hello_world_client.cpp

Chapter 2. What’s so special about HPX?
// Copyright (c) 2012 Bryce Lelbach
//
// SPDX-License-Identifier: BSL-1.0
// Distributed under the Boost Software License, Version 1.0. (See accompanying
// file LICENSE_1_0.txt or copy at http://www.boost.org/LICENSE_1_0.txt)

#include "hello_world_component.hpp"
#include <hpx/hpx_init.hpp>

int hpx_main(hpx::program_options::variables_map&)
{
    // Create a single instance of the component on this locality.
    examples::hello_world client = hpx::new_<examples::hello_world>(hpx::find_here());
    // Invoke the component's action, which will print "Hello World!".
    client.invoke();

    return hpx::finalize(); // Initiate shutdown of the runtime system.
}

int main(int argc, char* argv[])
{
    return hpx::init(argc, argv); // Initialize and run HPX.
}

Copy the three source files above into three files (called hello_world_component.cpp, hello_world_component.hpp and hello_world_client.cpp, respectively).

Now, in the directory where you put the files, run the following command to build the component library. (where $HPX_LOCATION is the build directory or CMAKE_INSTALL_PREFIX you used while building HPX):

```
export PKG_CONFIG_PATH=$PKG_CONFIG_PATH:$HPX_LOCATION/lib/pkgconfig
c++ -o libhpx_hello_world.so hello_world_component.cpp \
    `pkg-config --cflags --libs hpx_component` \
    -lhpx_iostreams -DHPX_COMPONENT_NAME=hpx_hello_world
```

Now pick a directory in which to install your HPX component libraries. For this example, we’ll choose a directory named my_hpex_libs:

```
mkdir ~/my_hpex_libs
mv libhpx_hello_world.so ~/my_hpex_libs
```

**Note:** HPX libraries have different names in debug and release mode. If you want to link against a debug HPX library, you need to use the _debug suffix for the pkg-config name. That means instead of hpx_application or hpx_component you will have to use hpx_application_debug or hpx_component_debug. Moreover, all referenced HPX components need to have a appended d suffix, e.g. instead of -lhpx_iostreams you will need to specify -lhpx_iostreamsd.

**Important:** If the HPX libraries are in a path that is not found by the dynamic linker. You need to add the path
$HPX_LOCATION/lib to your linker search path (for example LD_LIBRARY_PATH on Linux).

Now, to build the application that uses this component (hello_world_client.cpp), we do:

```bash
export PKG_CONFIG_PATH=$PKG_CONFIG_PATH:$HPX_LOCATION/lib/pkgconfig
c++ -o hello_world_client hello_world_client.cpp \
`pkg-config --cflags --libs hpx_application` \ 
-L$HOME/my_hpx_libs -lhpx_hello_world -lhpx_iostreams
```

**Important:** When using pkg-config with HPX, the pkg-config flags must go after the –o flag.

Finally, you’ll need to set your LD_LIBRARY_PATH before you can run the program. To run the program, type:

```bash
export LD_LIBRARY_PATH="$LD_LIBRARY_PATH:$HOME/my_hpx_libs"
./hello_world_client
```

which should print Hello HPX World! and exit.

**Using HPX with CMake-based projects**

In addition to the pkg-config support discussed on the previous pages, HPX comes with full CMake support. In order to integrate HPX into existing or new CMakeLists.txt, you can leverage the find_package command integrated into CMake. Following, is a Hello World component example using CMake.

Let’s revisit what we have. We have three files that compose our example application:

- hello_world_component.hpp
- hello_world_component.cpp
- hello_world_client.hpp

The basic structure to include HPX into your CMakeLists.txt is shown here:

```bash
# Require a recent version of cmake
cmake_minimum_required(VERSION 3.3.2 FATAL_ERROR)

# This project is C++ based.
project(your_app CXX)

# Instruct cmake to find the HPX settings
find_package(HPX)
```

In order to have CMake find HPX, it needs to be told where to look for the HPXConfig.cmake file that is generated when HPX is built or installed. It is used by find_package(HPX) to set up all the necessary macros needed to use HPX in your project. The ways to achieve this are:

- Set the HPX_DIR CMake variable to point to the directory containing the HPXConfig.cmake script on the command line when you invoke CMake:

  ```bash
cmake -DHPX_DIR=$HPX_LOCATION/lib/cmake/HPX ...
  ```

  where $HPX_LOCATION is the build directory or CMAKE_INSTALL_PREFIX you used when building/configuring HPX.

---

106 https://www.cmake.org/cmake/help/latest/command/find_package.html
• Set the CMAKE_PREFIX_PATH variable to the root directory of your HPX build or install location on the command line when you invoke CMake:

```cmake
   cmake -DCMAKE_PREFIX_PATH=$HPX_LOCATION ...
```

The difference between CMAKE_PREFIX_PATH and HPX_DIR is that CMake will add common postixes, such as lib/cmake/<project>, to the MAKE_PREFIX_PATH and search in these locations too. Note that if your project uses HPX as well as other CMake-managed projects, the paths to the locations of these multiple projects may be concatenated in the CMAKE_PREFIX_PATH.

• The variables above may be set in the CMake GUI or curses ccmake interface instead of the command line.

Additionally, if you wish to require HPX for your project, replace the find_package(HPX) line with find_package(HPX REQUIRED).

You can check if HPX was successfully found with the HPX_FOUND CMake variable.

The simplest way to add the HPX component is to use the add_hpx_component macro and add it to the CMakeLists.txt file:

```cmake
# build your application using HPX
add_hpx_component(hello_world
   SOURCES hello_world_component.cpp
   HEADERS hello_world_component.hpp
   COMPONENT_DEPENDENCIES iostreams)
```

**Note:** add_hpx_component adds a _component suffix to the target name. In the example above, a hello_world_component target will be created.

The available options to add_hpx_component are:

- **SOURCES:** The source files for that component
- **HEADERS:** The header files for that component
- **DEPENDENCIES:** Other libraries or targets this component depends on
- **COMPONENT_DEPENDENCIES:** The components this component depends on
- **PLUGIN:** Treats this component as a plugin-able library
- **COMPILE_FLAGS:** Additional compiler flags
- **LINK_FLAGS:** Additional linker flags
- **FOLDER:** Adds the headers and source files to this Source Group folder
- **EXCLUDE_FROM_ALL:** Do not build this component as part of the all target

After adding the component, the way you add the executable is as follows:

```cmake
# build your application using HPX
add_hpx_executable(hello_world
   ESSENTIAL
   SOURCES hello_world_client.cpp
   COMPONENT_DEPENDENCIES hello_world)
```

**Note:** add_hpx_executable automatically adds a _component suffix to dependencies specified in COMPONENT_DEPENDENCIES, meaning you can directly use the name given when adding a component using
add_hpx_component.

When you configure your application, all you need to do is set the HPX_DIR variable to point to the installation of HPX.

**Note:** All library targets built with HPX are exported and readily available to be used as arguments to target_link_libraries in your targets. The HPX include directories are available with the HPX_INCLUDE_DIRS CMake variable.

**CMake macros to integrate HPX into existing applications**

In addition to the add_hpx_component and add_hpx_executable, you can use the hpx_setup_target macro to have an already existing target to be used with the HPX libraries:

```
add_hpx_component (target)
```

Optional parameters are:

- **EXPORT:** Adds it to the CMake export list HPXTargets
- **INSTALL:** Generates an install rule for the target
- **PLUGIN:** Treats this component as a plugin-able library
- **TYPE:** The type can be: EXECUTABLE, LIBRARY or COMPONENT
- **DEPENDENCIES:** Other libraries or targets this component depends on
- **COMPONENT_DEPENDENCIES:** The components this component depends on
- **COMPILE_FLAGS:** Additional compiler flags
- **LINK_FLAGS:** Additional linker flags

If you do not use CMake, you can still build against HPX, but you should refer to the section on *How to build HPX components with pkg-config*.

**Note:** Since HPX relies on dynamic libraries, the dynamic linker needs to know where to look for them. If HPX isn’t installed into a path that is configured as a linker search path, external projects need to either set RPATH or adapt LD_LIBRARY_PATH to point to where the HPX libraries reside. In order to set RPATHs, you can include HPX_SetFullRPATH in your project after all libraries you want to link against have been added. Please also consult the CMake documentation here.

**Using HPX with Makefile**

A basic project building with HPX is through creating makefiles. The process of creating one can get complex depending upon the use of cmake parameter HPX_WITH_HPX_MAIN (which defaults to ON).

**How to build HPX applications with makefile**

If HPX is installed correctly, you should be able to build and run a simple Hello World program. It prints Hello World! on the locality you run it on.

---

107 https://www.cmake.org/cmake/help/latest/command/target_link_libraries.html
Copy the content of this program into a file called hello_world.cpp.

Now, in the directory where you put hello_world.cpp, create a Makefile. Add the following code:

```makefile
CXX=(CXX) # Add your favourite compiler here or let makefile choose default.
CXXFLAGS=-O3 -std=c++17
BOOST_ROOT=/path/to/boost
HWLOC_ROOT=/path/to/hwloc
TCMALLOC_ROOT=/path/to/tcmalloc
HPX_ROOT=/path/to/hpx

INCLUDE_DIRECTIVES=$($HPX_ROOT)/include $($BOOST_ROOT)/include $($HWLOC_ROOT)/include

LIBRARY_DIRECTIVES=-L$(HPX_ROOT)/lib $(HPX_ROOT)/lib/libhpx_init.a $(HPX_ROOT)/lib/libhpx.so $(BOOST_ROOT)/lib/libboost_atomic-mt.so $(BOOST_ROOT)/lib/libboost_system-mt.so $(BOOST_ROOT)/lib/libboost_regex-mt.so $(BOOST_ROOT)/lib/libboost_program_options-mt.so $(TCMALLOC_ROOT)/libtcmalloc_minimal.so $(HWLOC_ROOT)/libhwloc.so -lpthread $(TCMALLOC_ROOT)/libhpx.so

LINK_FLAGS=$($HPX_ROOT)/lib/libhpx_wrap.a -Wl,-wrap=main # should be left empty for HPX_WITH_HPX_MAIN=OFF

hello_world: hello_world.o
 $(CXX) $(CXXFLAGS) -o hello_world hello_world.o $(LIBRARY_DIRECTIVES) $(LINK_FLAGS)

hello_world.o:
 $(CXX) $(CXXFLAGS) -c -o hello_world.o hello_world.cpp $(INCLUDE_DIRECTIVES)

Important: LINK_FLAGS should be left empty if HPX_WITH_HPX_MAIN is set to OFF. Boost in the above example is build with --layout=tagged. Actual Boost flags may vary on your build of Boost.
To build the program, type:

```
make
```

A successful build should result in `hello_world` binary. To test, type:

```
./hello_world
```

**How to build HPX components with makefile**

Let’s try a more complex example involving an `HPX` component. An `HPX` component is a class that exposes `HPX` actions. `HPX` components are compiled into dynamically-loaded modules called component libraries. Here’s the source code:

**hello_world_component.cpp**

```cpp
#include "hello_world_component.hpp"
#include <hpx/include/iostreams.hpp>
#include <iostream>

namespace examples { namespace server {
    void hello_world::invoke()
    {
        hpx::cout << "Hello HPX World!" << std::endl;
    }
}}

HPX_REGISTER_COMPONENT_MODULE();

typedef hpx::components::component<
    examples::server::hello_world
> hello_world_type;

HPX_REGISTER_COMPONENT(hello_world_type, hello_world);

HPX_REGISTER_ACTION(

... (continues on next page)
```cpp
: hpx::components::component_base<hello_world>
{
  void invoke();
  HPX_DEFINE_COMPONENT_ACTION(hello_world, invoke);
};
}

HPX_REGISTER_ACTION_DECLARATION(
  examples::server::hello_world::invoke_action, hpx::hello_world_invoke_action);

namespace examples {

  struct hello_world : hpx::components::client_base<hello_world, server::hello_world>
  {
    typedef hpx::components::client_base<hello_world, server::hello_world> base_type;

    hello_world(hpx::future<hpx::naming::id_type> && f)
      : base_type(std::move(f))
    {}

    hello_world(hpx::naming::id_type && f)
      : base_type(std::move(f))
    {}

    void invoke()
    {
      hpx::async<server::hello_world::invoke_action>(
        this->get_id()).get();
    }

  };

}
```

### hello_world_client.cpp

```cpp
// Copyright (c) 2012 Bryce Lelbach

#include "hello_world_component.hpp"
#include <hpx/hpx_init.hpp>

int hpx_main(hpx::program_options::variables_map& vm)
{
  {
    // Create a single instance of the component on this locality.
    examples::hello_world client =
      hpx::new_<examples::hello_world>(hpx::find_here());

    // Invoke the component’s action, which will print “Hello World!”.
    client.invoke();
  }

  return hpx::finalize(); // Initiate shutdown of the runtime system.
```

(continues on next page)
Now, in the directory, create a Makefile. Add the following code:

```
CXX=(CXX)  # Add your favourite compiler here or let makefile choose default.
CXXFLAGS=-O3 -std=c++17
BOOST_ROOT=/path/to/boost
HWLOC_ROOT=/path/to/hwloc
TCMALLOC_ROOT=/path/to/tcmalloc
HPX_ROOT=/path/to/hpx

INCLUDE_DIRECTIVES=$(HPX_ROOT)/include $(BOOST_ROOT)/include $(HWLOC_ROOT)/include
LIBRARY_DIRECTIVES=-L$(HPX_ROOT)/lib $(HPX_ROOT)/lib/libhpx_init.a $(HPX_ROOT)/lib/libhpx
    filesystem-rt.so $(BOOST_ROOT)/lib/libboost_atomic-mt.so $(BOOST_ROOT)/lib/libboost_regex-mt.so $(BOOST_ROOT)/lib/libboost_system-mt.so -lpthread $(TCMALLOC_ROOT)/lib/libtcmalloc_minimal.so $(HWLOC_ROOT)/lib/libhwloc.so -ldl -lrt

LINK_FLAGS=$(HPX_ROOT)/lib/libhpx_wrap.a -Wl,-wrap=main # should be left empty if HPX_WITH_HPX_MAIN=OFF

hello_world_client: libhpx_hello_world hello_world_client.o
    $(CXX) $(CXXFLAGS) -o hello_world_client $(LIBRARY_DIRECTIVES) libhpx_hello_world
    $(LINK_FLAGS)

hello_world_client.o: hello_world_client.cpp
    $(CXX) $(CXXFLAGS) -o hello_world_client.o hello_world_client.cpp $(INCLUDE_DIRECTIVES)

libhpx_hello_world: hello_world_component.o
    $(CXX) $(CXXFLAGS) -o libhpx_hello_world hello_world_component.o $(LIBRARY_DIRECTIVES)

hello_world_component.o: hello_world_component.cpp
    $(CXX) $(CXXFLAGS) -c -o hello_world_component.o hello_world_component.cpp
    $(INCLUDE DIRECTIVES)
```

To build the program, type:

```
make
```

A successfull build should result in hello_world binary. To test, type:

```
./hello_world
```

**Note:** Due to high variations in CMake flags and library dependencies, it is recommended to build HPX applications and components with pkg-config or CMakeLists.txt. Writing Makefile may result in broken builds if due care is not taken.
pkg-config files and CMake systems are configured with CMake build of HPX. Hence, they are stable when used together and provide better support overall.

2.5.4 Starting the HPX runtime

In order to write an application which uses services from the HPX runtime system you need to initialize the HPX library by inserting certain calls into the code of your application. Depending on your use case, this can be done in 3 different ways:

- **Minimally invasive**: Re-use the main() function as the main HPX entry point.
- **Balanced use case**: Supply your own main HPX entry point while blocking the main thread.
- **Most flexibility**: Supply your own main HPX entry point while avoiding to block the main thread.
- **Suspend and resume**: As above but suspend and resume the HPX runtime to allow for other runtimes to be used.

**Re-use the main() function as the main HPX entry point**

This method is the least intrusive to your code. It however provides you with the smallest flexibility in terms of initializing the HPX runtime system. The following code snippet shows what a minimal HPX application using this technique looks like:

```cpp
#include <hpx/hpx_main.hpp>

int main(int argc, char* argv[])
{
    return 0;
}
```

The only change to your code you have to make is to include the file `hpx/hpx_main.hpp`. In this case the function main() will be invoked as the first HPX thread of the application. The runtime system will be initialized behind the scenes before the function main() is executed and will automatically stop after main() has returned. All HPX API functions can be used from within this function now.

**Note:** The function main() does not need to expect receiving argc argv as shown above, but could expose the signature int main(). This is consistent with the usually allowed prototypes for the function main() in C++ applications.

All command line arguments specific to HPX will still be processed by the HPX runtime system as usual. However, those command line options will be removed from the list of values passed to argc/argv of the function main(). The list of values passed to main() will hold only the commandline options which are not recognized by the HPX runtime system (see the section HPX Command Line Options for more details on what options are recognized by HPX).

**Note:** In this mode all one-letter-shortcuts are disabled which are normally available on the HPX command line (such as -t or -l see HPX Command Line Options). This is done to minimize any possible interaction between the command line options recognized by the HPX runtime system and any command line options defined by the application.

The value returned from the function main() as shown above will be returned to the operating system as usual.

**Important:** To achieve this seamless integration, the header file `hpx/hpx_main.hpp` defines a macro:
#define main hpx_startup::user_main

which could result in unexpected behavior.

**Important:** To achieve this seamless integration, we use different implementations for different Operating Systems. In case of Linux or Mac OSX, the code present in hpx_wrap.cpp is put into action. We hook into the system function in case of Linux and provide alternate entry point in case of Mac OSX. For other Operating Systems we rely on a macro:

#define main hpx_startup::user_main

provided in the header file hpx/hpx_main.hpp. This implementation can result in unexpected behavior.

**Caution:** We make use of an override variable include_libhpx_wrap in the header file hpx/hpx_main.hpp to swiftly choose the function call stack at runtime. Therefore, the header file should only be included in the main executable. Including it in the components will result in multiple definition of the variable.

### Supply your own main HPX entry point while blocking the main thread

With this method you need to provide an explicit main thread function named hpx_main at global scope. This function will be invoked as the main entry point of your HPX application on the console locality only (this function will be invoked as the first HPX thread of your application). All HPX API functions can be used from within this function.

The thread executing the function hpx::init will block waiting for the runtime system to exit. The value returned from hpx_main will be returned from hpx::init after the runtime system has stopped.

The function hpx::finalize has to be called on one of the HPX localities in order to signal that all work has been scheduled and the runtime system should be stopped after the scheduled work has been executed.

This method of invoking HPX has the advantage of you being able to decide which version of hpx::init to call. This allows to pass additional configuration parameters while initializing the HPX runtime system.

```cpp
#include <hpx/hpx_init.hpp>

int hpx_main(int argc, char* argv[])
{
    // Any HPX application logic goes here...
    return hpx::finalize();
}

int main(int argc, char* argv[])
{
    // Initialize HPX, run hpx_main as the first HPX thread, and
    // wait for hpx::finalize being called.
    return hpx::init(argc, argv);
}
```

**Note:** The function hpx_main does not need to expect receiving argc/argv as shown above, but could expose one of the following signatures:
int hpx_main();
int hpx_main(int argc, char* argv[]);
int hpx_main(hpx::program_options::variables_map& vm);

This is consistent with (and extends) the usually allowed prototypes for the function main() in C++ applications.

The header file to include for this method of using HPX is hpx/hpx_init.hpp.

There are many additional overloads of hpx::init available, such as for instance to provide your own entry point function instead of hpx_main. Please refer to the function documentation for more details (see: hpx/hpx_init.hpp).

**Supply your own main HPX entry point while avoiding to block the main thread**

With this method you need to provide an explicit main thread function named hpx_main at global scope. This function will be invoked as the main entry point of your HPX application on the console locality only (this function will be invoked as the first HPX thread of your application). All HPX API functions can be used from within this function.

The thread executing the function hpx::start will **not** block waiting for the runtime system to exit, but will return immediately.

**Important:** You cannot use any of the HPX API functions other that hpx::stop from inside your main() function.

The function hpx::finalize has to be called on one of the HPX localities in order to signal that all work has been scheduled and the runtime system should be stopped after the scheduled work has been executed.

This method of invoking HPX is useful for applications where the main thread is used for special operations, such a GUIs. The function hpx::stop can be used to wait for the HPX runtime system to exit and should be at least used as the last function called in main(). The value returned from hpx_main will be returned from hpx::stop after the runtime system has stopped.

```cpp
#include <hpx/hpx_start.hpp>

int hpx_main(int argc, char* argv[])
{
    // Any HPX application logic goes here...
    return hpx::finalize();
}

int main(int argc, char* argv[])
{
    // Initialize HPX, run hpx_main.
    hpx::start(argc, argv);

    // ...Execute other code here...

    // Wait for hpx::finalize being called.
    return hpx::stop();
}
```

**Note:** The function hpx_main does not need to expect receiving argc/argv as shown above, but could expose one of the following signatures:

2.5. Manual
int hpx_main();
int hpx_main(int argc, char* argv[]);
int hpx_main(hpx::program_options::variables_map& vm);

This is consistent with (and extends) the usually allowed prototypes for the function main() in C++ applications.

The header file to include for this method of using HPX is hpx/hpx_start.hpp.

There are many additional overloads of hpx::start available, such as for instance to provide your own entry point function instead of hpx_main. Please refer to the function documentation for more details (see: hpx/hpx_start.hpp).

**Suspending and resuming the HPX runtime**

In some applications it is required to combine HPX with other runtimes. To support this use case HPX provides two functions: hpx::suspend and hpx::resume. hpx::suspend is a blocking call which will wait for all scheduled tasks to finish executing and then put the thread pool OS threads to sleep. hpx::resume simply wakes up the sleeping threads so that they are ready to accept new work. hpx::suspend and hpx::resume can be found in the header hpx/hpx_suspend.hpp.

```cpp
#include <hpx/hpx_start.hpp>
#include <hpx/hpx_suspend.hpp>

int main(int argc, char* argv[]) {

    // Initialize HPX, don't run hpx_main
    hpx::start(nullptr, argc, argv);

    // Schedule a function on the HPX runtime
    hpx::apply(&my_function, ...);

    // Wait for all tasks to finish, and suspend the HPX runtime
    hpx::suspend();

    // Execute non-HPX code here

    // Resume the HPX runtime
    hpx::resume();

    // Schedule more work on the HPX runtime

    // hpx::finalize has to be called from the HPX runtime before hpx::stop
    hpx::apply([]() { hpx::finalize(); });
    return hpx::stop();
}
```

**Note:** hpx::suspend does not wait for hpx::finalize to be called. Only call hpx::finalize when you wish to fully stop the HPX runtime.

HPX also supports suspending individual thread pools and threads. For details on how to do that see the documentation for hpx::threads::thread_pool_base.
Automatically suspending worker threads

The previous method guarantees that the worker threads are suspended when you ask for it and that they stay suspended. An alternative way to achieve the same effect is to tweak how quickly HPX suspends its worker threads when they run out of work. The following configuration values make sure that HPX idles very quickly:

```
hpx.max_idle_backoff_time = 1000
hpx.max_idle_loop_count = 0
```

They can be set on the command line using `--hpx:ini=hpx.max_idle_backoff_time=1000` and `--hpx:ini=hpx.max_idle_loop_count=0`. See *Launching and configuring HPX applications* for more details on how to set configuration parameters.

After setting idling parameters the previous example could now be written like this instead:

```cpp
#include <hpx/hpx_start.hpp>

int main(int argc, char* argv[])
{
    // Initialize HPX, don't run hpx_main
    hpx::start(nullptr, argc, argv);

    // Schedule some functions on the HPX runtime
    // NOTE: run_as_hpx_thread blocks until completion.
    hpx::run_as_hpx_thread(&my_function, ...);
    hpx::run_as_hpx_thread(&my_other_function, ...);

    // hpx::finalize has to be called from the HPX runtime before hpx::stop
    hpx::apply([]() { hpx::finalize(); });
    return hpx::stop();
}
```

In this example each call to `hpx::run_as_hpx_thread` acts as a “parallel region”.

Working of *hpx_main.hpp*

In order to initialize HPX from `main()`, we make use of linker tricks.

It is implemented differently for different Operating Systems. Method of implementation is as follows:

- **Linux**: Using linker `--wrap` option.
- **Mac OSX**: Using the linker `-e` option.
- **Windows**: Using `#define main hpx_startup::user_main`

Linux implementation

We make use of the Linux linker ld’s `--wrap` option to wrap the `main()` function. This way any call to `main()` are redirected to our own implementation of main. It is here that we check for the existence of `hpx_main.hpp` by making use of a shadow variable `include_libhpx_wrap`. The value of this variable determines the function stack at runtime.

The implementation can be found in `libhpx_wrap.a`.

**Important**: It is necessary that `hpx_main.hpp` be not included more than once. Multiple inclusions can result in multiple definition of `include_libhpx_wrap`. 

---

2.5. Manual
Mac OS X implementation

Here we make use of yet another linker option \(-e\) to change the entry point to our custom entry function \(\text{initialize}\_main\). We initialize the \(\text{HPX}\) runtime system from this function and call main from the initialized system. We determine the function stack at runtime by making use of the shadow variable \(\text{include}\_\text{libhpx}\_\text{wrap}\).

The implementation can be found in \(\text{libhpx}\_\text{wrap}.a\).

**Important:** It is necessary that \(\text{hpx}\_\text{main}\.hpp\) be not included more than once. Multiple inclusions can result in multiple definition of \(\text{include}\_\text{libhpx}\_\text{wrap}\).

Windows implementation

We make use of a macro \#define main hpx\_startup::user\_main to take care of the initializations.

This implementation could result in unexpected behaviors.

2.5.5 Launching and configuring \(\text{HPX}\) applications

Configuring \(\text{HPX}\) applications

All \(\text{HPX}\) applications can be configured using special command line options and/or using special configuration files. This section describes the available options, the configuration file format, and the algorithm used to locate possible predefined configuration files. Additionally this section describes the defaults assumed if no external configuration information is supplied.

During startup any \(\text{HPX}\) application applies a predefined search pattern to locate one or more configuration files. All found files will be read and merged in the sequence they are found into one single internal database holding all configuration properties. This database is used during the execution of the application to configure different aspects of the runtime system.

In addition to the ini files, any application can supply its own configuration files, which will be merged with the configuration database as well. Moreover, the user can specify additional configuration parameters on the command line when executing an application. The \(\text{HPX}\) runtime system will merge all command line configuration options (see the description of the \(--\text{hpx:ini}, --\text{hpx:config}, \text{and } --\text{hpx:app-config}\) command line options).

The \(\text{HPX}\) INI File Format

All \(\text{HPX}\) applications can be configured using a special file format which is similar to the well-known Windows INI file format\(^{109}\). This is a structured text format allowing to group key/value pairs (properties) into sections. The basic element contained in an ini file is the property. Every property has a name and a value, delimited by an equals sign `=`. The name appears to the left of the equals sign:

```
name=value
```

The value may contain equal signs as only the first `=` character is interpreted as the delimiter between \textit{name} and \textit{value}. Whitespace before the name, after the value and immediately before and after the delimiting equal sign is ignored. Whitespace inside the value is retained.

Properties may be grouped into arbitrarily named sections. The section name appears on a line by itself, in square brackets [ and ]. All properties after the section declaration are associated with that section. There is no explicit “end of section” delimiter; sections end at the next section declaration, or the end of the file:

\(^{109}\) https://en.wikipedia.org/wiki/INI_file
In *HPX* sections can be nested. A nested section has a name composed of all section names it is embedded in. The section names are concatenated using a dot `. `:

```
[outer_section.inner_section]
```

Here `inner_section` is logically nested within `outer_section`.

It is possible to use the full section name concatenated with the property name to refer to a particular property. For example in:

```
[a.b.c]
d = e
```

the property value of `d` can be referred to as `a.b.c.d=e`.

In *HPX* ini files can contain comments. Hash signs `#` at the beginning of a line indicate a comment. All characters starting with the `#` until the end of line are ignored.

If a property with the same name is reused inside a section, the second occurrence of this property name will override the first occurrence (discard the first value). Duplicate sections simply merge their properties together, as if they occurred contiguously.

In *HPX* ini files, a property value `${FOO:default}` will use the environmental variable `FOO` to extract the actual value if it is set and `default` otherwise. No default has to be specified. Therefore `${FOO}` refers to the environmental variable `FOO`. If `FOO` is not set or empty the overall expression will evaluate to an empty string. A property value `${[section.key:default]` refers to the value held by the property `section.key` if it exists and `default` otherwise. No default has to be specified. Therefore `${[section.key]` refers to the property `section.key`. If the property `section.key` is not set or empty, the overall expression will evaluate to an empty string.

**Note:** Any property `${[section.key:default]` is evaluated whenever it is queried and not when the configuration data is initialized. This allows for lazy evaluation and relaxes initialization order of different sections. The only exception are recursive property values, e.g. values referring to the very key they are associated with. Those property values are evaluated at initialization time to avoid infinite recursion.

**Built-in Default Configuration Settings**

During startup any *HPX* application applies a predefined search pattern to locate one or more configuration files. All found files will be read and merged in the sequence they are found into one single internal data structure holding all configuration properties.

As a first step the internal configuration database is filled with a set of default configuration properties. Those settings are described on a section by section basis below.

**Note:** You can print the default configuration settings used for an executable by specifying the command line option `--hpx:dump-config`.

**The system configuration section**

```
[system]
pid = <process-id>
```

(continues on next page)
The `hpx` configuration section

```plaintext
[hpx]
location = ${HPX_LOCATION:$[system.prefix]}
component_path = ${hpx.location}/lib/hpx:${system.executable_prefix}/lib/hpx:${system.executable_prefix}/../lib/hpx
master_ini_path = ${hpx.location}/share/hpx-<version>:${system.executable_prefix}/share/hpx-<version>:${system.executable_prefix}/../share/hpx-<version>
ini_path = ${hpx.master_ini_path}/ini
os_threads = 1
localities = 1
program_name =
cmd_line =
lock_detection = ${HPX_LOCK_DETECTION:0}
throw_on_held_lock = ${HPX_THROW_ON_HELD_LOCK:1}
minimal_deadlock_detection = <debug>
spinlock_deadlock_detection = <debug>
spinlock_deadlock_detection_limit = ${HPX_SPINLOCK_DEADLOCK_DETECTION_LIMIT:1000000}
max_background_threads = ${HPX_MAX_BACKGROUND_THREADS:$[hpx.os_threads]}
max_idle_loop_count = ${HPX_MAX_IDLE_LOOP_COUNT:<hpx_idle_loop_count_max>}
max_busy_loop_count = ${HPX_MAX_BUSY_LOOP_COUNT:<hpx_busy_loop_count_max>}
max_idle_backoff_time = ${HPX_MAX_IDLE_BACKOFF_TIME:<hpx_idle_backoff_time_max>}

[hpx.stacks]
small_size = ${HPX_SMALL_STACK_SIZE:<hpx_small_stack_size>}
medium_size = ${HPX_MEDIUM_STACK_SIZE:<hpx_medium_stack_size>}
large_size = ${HPX_LARGE_STACK_SIZE:<hpx_large_stack_size>}
huge_size = ${HPX_HUGE_STACK_SIZE:<hpx_huge_stack_size>}
use_guard_pages = ${HPX_THREAD_GUARD_PAGE:1}
```

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>system.pid</td>
<td>This is initialized to store the current OS-process id of the application instance.</td>
</tr>
<tr>
<td>system.prefix</td>
<td>This is initialized to the base directory <code>HPX</code> has been loaded from.</td>
</tr>
<tr>
<td>system.executable_prefix</td>
<td>This is initialized to the base directory the current executable has been loaded from.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>hpx.location</code></td>
<td>This is initialized to the id of the <code>locality</code> this application instance is running on.</td>
</tr>
<tr>
<td><code>hpx.component</code></td>
<td>Duplicates are discarded. This property can refer to a list of directories separated by <code>:</code> (Linux, Android, and MacOS) or using <code>;</code> (Windows).</td>
</tr>
<tr>
<td><code>hpx.master_ini</code></td>
<td>This is initialized to the list of default paths of the main <code>hpx.ini</code> configuration files. This property can refer to a list of directories separated by <code>:</code> (Linux, Android, and MacOS) or using <code>;</code> (Windows).</td>
</tr>
<tr>
<td><code>hpx.ini_path</code></td>
<td>This is initialized to the default path where <code>HPX</code> will look for more ini configuration files. This property can refer to a list of directories separated by <code>:</code> (Linux, Android, and MacOS) or using <code>;</code> (Windows).</td>
</tr>
<tr>
<td><code>hpx.os_threads</code></td>
<td>This setting reflects the number of OS-threads used for running <code>HPX</code>-threads. Defaults to number of detected cores (not hyperthreads/PUs).</td>
</tr>
<tr>
<td><code>hpx.localities</code></td>
<td>This setting reflects the number of localities the application is running on. Defaults to 1.</td>
</tr>
<tr>
<td><code>hpx.program_name</code></td>
<td>This setting reflects the program name of the application instance. Initialized from the command line. argv[0].</td>
</tr>
<tr>
<td><code>hpx.cmd_line</code></td>
<td>This setting reflects the actual command line used to launch this application instance.</td>
</tr>
<tr>
<td><code>hpx.lock_detection</code></td>
<td>This setting verifies that no locks are being held while a <code>HPX</code> thread is suspended. This setting is applicable only if <code>HPX_WITH_VERIFY_LOCKS</code> is set during configuration in CMake.</td>
</tr>
<tr>
<td><code>hpx.throw_on_held</code></td>
<td>This setting causes an exception if during lock detection at least one lock is being held while a <code>HPX</code> thread is suspended. This setting is applicable only if <code>HPX_WITH_VERIFY_LOCKS</code> is set during configuration in CMake. This setting has no effect if <code>hpx.lock_detection=0</code>.</td>
</tr>
<tr>
<td><code>hpx.minimal_deadlock_detection_limit</code></td>
<td>This setting enables support for minimal deadlock detection for <code>HPX</code>-threads. By default this is set to 0 (for Debug builds) or to 0 (for Release, RelWithDebInfo, RelMinSize builds), this setting is effective only if <code>HPX_WITH_THREAD_DEADLOCK_DETECTION</code> is set during configuration in CMake.</td>
</tr>
<tr>
<td><code>hpx.minimal_deadlock_detection_limit</code></td>
<td>This setting verifies that spinlocks don’t spin longer than specified using the <code>hpx.spinlockDeadlineDetectionLimit</code>. This setting is applicable only if <code>HPX_WITH_SPINLOCK_DEADLOCK_DETECTION</code> is set during configuration in CMake. By default this is set to 1 (for Debug builds) or to 0 (for Release, RelWithDebInfo, RelMinSize builds).</td>
</tr>
<tr>
<td><code>hpx.spinlock_deadline_detection_limit</code></td>
<td>This setting specifies the upper limit of allowed number of spins that spinlocks are allowed to perform. This setting is applicable only if <code>HPX_WITH_SPINLOCK_DEADLOCK_DETECTION</code> is set during configuration in CMake. By default this is set to 1000000.</td>
</tr>
<tr>
<td><code>hpx.max_background_threads</code></td>
<td>This setting defines the number of threads in the scheduler which are used to execute background work. By default this is the same as the number of cores used for the scheduler.</td>
</tr>
<tr>
<td><code>hpx.max_idle_loop</code></td>
<td>This setting defines the maximum value of the busy-loop counter in the scheduler. By default this is defined by the preprocessor constant <code>HPX_BUSY_LOOP_COUNT_MAX</code>. This is an internal setting which you should change <strong>only if you know exactly what you are doing</strong>.</td>
</tr>
<tr>
<td><code>hpx.max_busy_loop</code></td>
<td>This setting defines the maximum time (in milliseconds) for the scheduler to sleep after belonging idle for the <code>hpx.max_idle_loop_count</code> iterations. This setting is applicable only if <code>HPX_WITH_THREAD_MANAGER_IDLE_BACKOFF</code> is set during configuration in CMake. By default this is defined by the preprocessor constant <code>HPX_IDLE_BACKOFF_TIME_MAX</code>. This is an internal setting which you should change <strong>only if you know exactly what you are doing</strong>.</td>
</tr>
<tr>
<td><code>hpx.stacks.small_size</code></td>
<td>This is initialized to the small stack size to be used by <code>HPX</code>-threads. Set by default to the value of the compile time preprocessor constant <code>HPX_SMALL_STACK_SIZE</code> (defaults to 0x8000). This value is used for all <code>HPX</code> threads by default, except for the thread running <code>hpx_main</code> (which runs on a large stack).</td>
</tr>
<tr>
<td><code>hpx.stacks.large_size</code></td>
<td>This is initialized to the medium stack size to be used by <code>HPX</code>-threads. Set by default to the value of the compile time preprocessor constant <code>HPX_MEDIUM_STACK_SIZE</code> (defaults to 0x20000).</td>
</tr>
<tr>
<td><code>hpx.stacks.huge_size</code></td>
<td>This is initialized to the large stack size to be used by <code>HPX</code>-threads. Set by default to the value of the compile time preprocessor constant <code>HPX_LARGE_STACK_SIZE</code> (defaults to 0x200000).</td>
</tr>
<tr>
<td><code>hpx.stacks.huge_size</code></td>
<td>This is initialized to the huge stack size to be used by <code>HPX</code>-threads. Set by default to the value of the compile time preprocessor constant <code>HPX_HUGE_STACK_SIZE</code> (defaults to 0x1000000).</td>
</tr>
</tbody>
</table>
The `hpx.threadpools` configuration section

```
[hpthreadpools]
io_pool_size = ${HPX_NUM_IO_POOL_SIZE:2}
parcel_pool_size = ${HPX_NUM_PARCEL_POOL_SIZE:2}
timer_pool_size = ${HPX_NUM_TIMER_POOL_SIZE:2}
```

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hpx.threadpools.io_pool_size</td>
<td>The value of this property defines the number of OS-threads created for the internal I/O thread pool.</td>
</tr>
<tr>
<td>hpx.threadpools.parcel_pool_size</td>
<td>The value of this property defines the number of OS-threads created for the internal parcel thread pool.</td>
</tr>
<tr>
<td>hpx.threadpools.timer_pool_size</td>
<td>The value of this property defines the number of OS-threads created for the internal timer thread pool.</td>
</tr>
</tbody>
</table>

The `hpx.thread_queue` configuration section

**Important:** These settings control internal values used by the thread scheduling queues in the HPX scheduler. You should not modify these settings except if you know exactly what you are doing!

```
[hpx.thread_queue]
min_tasks_to_steal_pending = ${HPX_THREAD_QUEUE_MIN_TASKS_TO_STEAL_PENDING:0}
min_tasks_to_steal_staged = ${HPX_THREAD_QUEUE_MIN_TASKS_TO_STEAL_STAGED:10}
min_add_new_count = ${HPX_THREAD_QUEUE_MIN_ADD_NEW_COUNT:10}
max_add_new_count = ${HPX_THREAD_QUEUE_MAX_ADD_NEW_COUNT:10}
max_delete_count = ${HPX_THREAD_QUEUE_MAX_DELETE_COUNT:1000}
```

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hpx.thread_queue.min_tasks_to_steal_pending</td>
<td>The value of this property defines the number of pending HPX threads which have to be available before neighboring cores are allowed to steal work. The default is to allow stealing always.</td>
</tr>
<tr>
<td>hpx.thread_queue.min_tasks_to_steal_staged</td>
<td>The value of this property defines the number of staged HPX tasks which have to be available before neighboring cores are allowed to steal work. The default is to allow stealing only if there are more than 10 tasks available.</td>
</tr>
<tr>
<td>hpx.thread_queue.min_add_new_count</td>
<td>The value of this property defines the minimal number of tasks to be converted into HPX threads whenever the thread queues for a core have run empty.</td>
</tr>
<tr>
<td>hpx.thread_queue.max_add_new_count</td>
<td>The value of this property defines the maximal number of tasks to be converted into HPX threads whenever the thread queues for a core have run empty.</td>
</tr>
<tr>
<td>hpx.thread_queue.max_delete_count</td>
<td>The value of this property defines the number of terminated HPX threads to discard during each invocation of the corresponding function.</td>
</tr>
</tbody>
</table>

The `hpx.components` configuration section

```
[hpx.components]
load_external = ${HPX_LOAD_EXTERNAL_COMPONENTS:1}
```
This entry defines whether external components will be loaded on this *locality*. This entry normally is set to 1 and usually there is no need to directly change this value. It is automatically set to 0 for a dedicated AGAS server *locality*.

Additionally, the section `hpx.components` will be populated with the information gathered from all found components. The information loaded for each of the components will contain at least the following properties:

```
[hpx.components.<component_instance_name>]
name = <component_name>
path = <full_path_of_the_component_module>
enabled = ${hpx.components.load_external}
```

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>name</code></td>
<td>This is the name of a component, usually the same as the second argument to the macro used while registering the component with <code>HPX_REGISTER_COMPONENT</code>. Set by the component factory.</td>
</tr>
<tr>
<td><code>path</code></td>
<td>This is either the full path file name of the component module or the directory the component module is located in. In this case, the component module name will be derived from the property <code>name</code>. Set by the component factory.</td>
</tr>
<tr>
<td><code>enabled</code></td>
<td>This setting explicitly enables or disables the component. This is an optional property, <code>HPX</code> assumed that the component is enabled if it is not defined.</td>
</tr>
</tbody>
</table>

The value for `<component_instance_name>` is usually the same as for the corresponding `name` property. However generally it can be defined to any arbitrary instance name. It is used to distinguish between different ini sections, one for each component.

**The `hpx.parcel` configuration section**

```
[hpx.parcel]
address = ${HPX_PARCEL_SERVER_ADDRESS:hpex_initial_ip_address}
port = ${HPX_PARCEL_SERVER_PORT:hpex_initial_ip_port}
bootstrap = ${HPX_PARCEL_BOOTSTRAP:hpex_parcel_bootstrap}
max_connections = ${HPX_PARCEL_MAX_CONNECTIONS:hpex_parcel_max_connections}
max_connections_per_locality = ${HPX_PARCEL_MAX_CONNECTIONS_PER_LOCALITY:hpex_parcel_max_connections_per_locality}
max_message_size = ${HPX_PARCEL_MAX_MESSAGE_SIZE:hpex_parcel_max_message_size}
max_outbound_message_size = ${HPX_PARCEL_MAX_OUTBOUND_MESSAGE_SIZE:hpex_parcel_max_outbound_message_size}
array_optimization = ${HPX_PARCEL_ARRAY_OPTIMIZATION:1}
zero_copy_optimization = ${HPX_PARCEL_ZERO_COPY_OPTIMIZATION:${hpx.parcel.array_optimization}}
async_serialization = ${HPX_PARCEL_ASYNC_SERIALIZATION:1}
message_handlers = ${HPX_PARCEL_MESSAGE_HANDLERS:0}
```
<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hpx. parcel. address</td>
<td>This property defines the default IP address to be used for the parcel layer to listen to. This IP address will be used as long as no other values are specified (for instance using the --hpx:hp command line option). The expected format is any valid IP address or domain name format which can be resolved into an IP address. The default depends on the compile time preprocessor constant HPX_INITIAL_IP_ADDRESS(&quot;127.0.0.1&quot;).</td>
</tr>
<tr>
<td>hpx. parcel. port</td>
<td>This property defines the default IP port to be used for the parcel layer to listen to. This IP port will be used as long as no other values are specified (for instance using the --hpx:hp command line option). The default depends on the compile time preprocessor constant HPX_INITIAL_IP_PORT(7910).</td>
</tr>
<tr>
<td>hpx. parcel. bootstrap</td>
<td>This property defines which parcelport type should be used during application bootstrap. The default depends on the compile time preprocessor constant HPX_PARCEL_BOOTSTRAP(&quot;tcp&quot;).</td>
</tr>
<tr>
<td>hpx. parcel. max_connections</td>
<td>This property defines how many network connections between different localities are overall kept alive by each of locality. The default depends on the compile time preprocessor constant HPX_PARCEL_MAX_CONNECTIONS(512).</td>
</tr>
<tr>
<td>hpx. parcel. max_connections_per_locality</td>
<td>This property defines the maximum number of network connections that one locality will open to another locality. The default depends on the compile time preprocessor constant HPX_PARCEL_MAX_CONNECTIONS_PER_LOCALITY(4).</td>
</tr>
<tr>
<td>hpx. parcel. max_message_size</td>
<td>This property defines the maximum allowed message size which will be transferrable through the parcel layer. The default depends on the compile time preprocessor constant HPX_PARCEL_MAX_MESSAGE_SIZE(1000000000 bytes).</td>
</tr>
<tr>
<td>hpx. parcel. max_outbound_message_size</td>
<td>This property defines the maximum allowed outbound coalesced message size which will be transferrable through the parcel layer. The default depends on the compile time preprocessor constant HPX_PARCEL_MAX_OUTBOUND_MESSAGE_SIZE(1000000 bytes).</td>
</tr>
<tr>
<td>hpx. parcel. array_optimization</td>
<td>This property defines whether this locality is allowed to utilize array optimizations during serialization of parcel data. The default is 1.</td>
</tr>
<tr>
<td>hpx. parcel. zero_copy_optimization</td>
<td>This property defines whether this locality is allowed to utilize zero copy optimizations during serialization of parcel data. The default is the same value as set for hpx.parcel.array_optimization.</td>
</tr>
<tr>
<td>hpx. parcel. async_serialization</td>
<td>This property defines whether this locality is allowed to spawn a new thread for serialization (this is both for encoding and decoding parcels). The default is 1.</td>
</tr>
<tr>
<td>hpx. parcel. message_handlers</td>
<td>This property defines whether message handlers are loaded. The default is 0.</td>
</tr>
</tbody>
</table>

The following settings relate to the TCP/IP parcelport.

```bash
[hpx.parcel.tcp]
enable = ${HPX_HAVE_PARCELPORT_TCP:${hpx.parcel.enabled}}
array_optimization = ${HPX_PARCEL_TCP_ARRAY_OPTIMIZATION:${hpx.parcel.array_optimization}}
zero_copy_optimization = ${HPX_PARCEL_TCP_ZERO_COPY_OPTIMIZATION:${hpx.parcel.zero_copy_optimization}}
async_serialization = ${HPX_PARCEL_TCP_ASYNC_SERIALIZATION:${hpx.parcel.async_serialization}}
parcel_pool_size = ${HPX_PARCEL_TCP_PARCEL_POOL_SIZE:${hpx.threadpools.parcel_pool_size}}
max_connections = ${HPX_PARCEL_TCP_MAX_CONNECTIONS:${hpx.parcel.max_connections}}
max_connections_per_locality = ${HPX_PARCEL_TCP_MAX_CONNECTIONS_PER_LOCALITY:${hpx.parcel.max_connections_per_locality}}
```

(continues on next page)
max_message_size = ${HPX_PARCEL_TCP_MAX_MESSAGE_SIZE:$[hpx.parcel.max_message_size]}
max_outbound_message_size = ${HPX_PARCEL_TCP_MAX_OUTBOUND_MESSAGE_SIZE:$[hpx.parcel.
˓→max_outbound_message_size]}

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hpx.parcel.tcp.enable</td>
<td>Enable the use of the default TCP parcelport. Note that the initial bootstrap of the overall HPX application will be performed using the default TCP connections. This parcelport is enabled by default. This will be disabled only if MPI is enabled (see below).</td>
</tr>
<tr>
<td>hpx.parcel.tcp.array_optimization</td>
<td>This property defines whether this locality is allowed to utilize array optimizations in the TCP/IP parcelport during serialization of parcel data. The default is the same value as set for hpx.parcel.array_optimization.</td>
</tr>
<tr>
<td>hpx.parcel.tcp.zero_copy_optimization</td>
<td>This property defines whether this locality is allowed to utilize zero copy optimizations in the TCP/IP parcelport during serialization of parcel data. The default is the same value as set for hpx.parcel.zero_copy_optimization.</td>
</tr>
<tr>
<td>hpx.parcel.tcp.async_serialization</td>
<td>This property defines whether this locality is allowed to spawn a new thread for serialization in the TCP/IP parcelport (this is both for encoding and decoding parcels). The default is the same value as set for hpx.parcel.async_serialization.</td>
</tr>
<tr>
<td>hpx.parcel.tcp.parcel_pool_size</td>
<td>The value of this property defines the number of OS-threads created for the internal parcel thread pool of the TCP parcel port. The default is taken from hpx.threadpools.parcel_pool_size.</td>
</tr>
<tr>
<td>hpx.parcel.tcp.max_connections</td>
<td>This property defines how many network connections between different localities are overall kept alive by each of locality. The default is taken from hpx.parcel.max_connections.</td>
</tr>
<tr>
<td>hpx.parcel_tcp.max_connections_per_locality</td>
<td>This property defines the maximum number of network connections that one locality will open to another locality. The default is taken from hpx.parcel_max_connections_per_locality.</td>
</tr>
<tr>
<td>hpx.parcel_tcp.max_message_size</td>
<td>This property defines the maximum allowed message size which will be transferrable through the parcel layer. The default is taken from hpx.parcel.max_message_size.</td>
</tr>
<tr>
<td>hpx.parcel_tcp.max_outbound_message_size</td>
<td>This property defines the maximum allowed outbound coalesced message size which will be transferrable through the parcel layer. The default is taken from hpx.parcel.max_outbound_message_size.</td>
</tr>
</tbody>
</table>

The following settings relate to the MPI parcelport. These settings take effect only if the compile time constant HPX_HAVE_PARCELPORT_MPI is set (the equivalent cmake variable is HPX_WITH_PARCELPORT_MPI and has to be set to ON).

```
[hpx.parcel.mpi]
enable = ${HPX_HAVE_PARCELPORT_MPI:$[hpx.parcel.enabled]}
env = ${HPX_HAVE_PARCELPORT_MPI_ENV:VM2_COMM_WORLD_RANK,PMI_RANKOMPI_COMM_WORLD_SIZE,˓→ALPS_APP_PE}
multithreaded = ${HPX_HAVE_PARCELPORT_MPI_MULTITHREADED:0}
rank = <MPI_rank>
processor_name = <MPI_processor_name>
array_optimization = ${HPX_HAVE_PARCEL_MPI_ARRAY_OPTIMIZATION:$[hpx.parcel.array_˓→optimization]}
zero_copy_optimization = ${HPX_HAVE_PARCEL_MPI_ZERO_COPY_OPTIMIZATION:$[hpx.parcel.˓→zero_copy_optimization]}
use_io_pool = ${HPX_HAVE_PARCEL_MPI_USE_IO_POOL:$1}
async_serialization = ${HPX_HAVE_PARCEL_MPI_ASYNC_SERIALIZATION:$[hpx.parcel.async_˓→serialization]}
parcel_pool_size = ${HPX_HAVE_PARCEL_MPI_PARCELPOOL_SIZE:$[hpx.threadpools.parcel_˓→pool_size]}
```

(continues on next page)
max_connections = ${HPX_HAVE_PARCEL_MPI_MAX_CONNECTIONS:${hpx.parcel.max_connections}}
max_connections_per_locality = ${HPX_HAVE_PARCEL_MPI_MAX_CONNECTIONS_PER_LOCALITY:${hpx.parcel.max_connections_per_locality}}
max_message_size = ${HPX_HAVE_PARCEL_MPI_MAX_MESSAGE_SIZE:${hpx.parcel.max_message_size}}
max_outbound_message_size = ${HPX_HAVE_PARCEL_MPI_MAX_OUTBOUND_MESSAGE_SIZE:${hpx.parcel.max_outbound_message_size}}

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hpx.parcel.mpi.enable</td>
<td>Enable the use of the MPI parcelport. HPX tries to detect if the application was started within a parallel MPI environment. If the detection was successful, the MPI parcelport is enabled by default. To explicitly disable the MPI parcelport, set to 0. Note that the initial bootstrap of the overall HPX application will be performed using MPI as well.</td>
</tr>
<tr>
<td>hpx.parcel.mpi.env</td>
<td>This property influences which environment variables (comma separated) will be analyzed to find out whether the application was invoked by MPI.</td>
</tr>
<tr>
<td>hpx.parcel.mpi.multithreaded</td>
<td>This property is used to determine what threading mode to use when initializing MPI. If this setting is 0 HPX will initialize MPI with MPI_THREAD_SINGLE if the value is not equal to 0 HPX will initialize MPI with MPI_THREAD_MULTI.</td>
</tr>
<tr>
<td>hpx.parcel.mpi.rank</td>
<td>This property will be initialized to the MPI rank of the locality.</td>
</tr>
<tr>
<td>hpx.parcel.mpi.processor_name</td>
<td>This property will be initialized to the MPI processor name of the locality.</td>
</tr>
<tr>
<td>hpx.parcel.mpi.array_optimization</td>
<td>This property defines whether this locality is allowed to utilize array optimizations in the MPI parcelport during serialization of parcel data. The default is the same value as set for hpx.parcel.array_optimization.</td>
</tr>
<tr>
<td>hpx.parcel.mpi.zero_copy_optimization</td>
<td>This property defines whether this locality is allowed to utilize zero copy optimizations in the MPI parcelport during serialization of parcel data. The default is the same value as set for hpx.parcel.zero_copy_optimization.</td>
</tr>
<tr>
<td>hpx.parcel.mpi.use_io_pool</td>
<td>This property can be set to run the progress thread inside of HPX threads instead of a separate thread pool. The default is 1.</td>
</tr>
<tr>
<td>hpx.parcel.mpi.async_serialization</td>
<td>This property defines whether this locality is allowed to spawn a new thread for serialization in the MPI parcelport (this is both for encoding and decoding parcels). The default is the same value as set for hpx.parcel.async_serialization.</td>
</tr>
<tr>
<td>hpx.parcel.mpi.pool_size</td>
<td>The value of this property defines the number of OS-threads created for the internal parcel thread pool of the MPI parcel port. The default is taken from hpx.threadpools.parcel_pool_size.</td>
</tr>
<tr>
<td>hpx.parcel.mpi.max_connections</td>
<td>This property defines how many network connections between different localities are overall kept alive by each of locality. The default is taken from hpx.parcel.max_connections.</td>
</tr>
<tr>
<td>hpx.parcel.mpi.max_connections_per_locality</td>
<td>This property defines the maximum number of network connections that one locality will open to another locality. The default is taken from hpx.parcel.max_connections_per_locality.</td>
</tr>
<tr>
<td>hpx.parcel.mpi.max_message_size</td>
<td>This property defines the maximum allowed message size which will be transferrable through the parcel layer. The default is taken from hpx.parcel.max_message_size.</td>
</tr>
<tr>
<td>hpx.parcel.mpi.max_outbound_message_size</td>
<td>This property defines the maximum allowed outbound coalesced message size which will be transferrable through the parcel layer. The default is taken from hpx.parcel.max_outbound_message_size.</td>
</tr>
</tbody>
</table>
The **hpx.agas** configuration section

```plaintext
[hpX.agas]
address = ${HPX_AGAS_SERVER_ADDRESS:<hpx_initial_ip_address>}
port = ${HPX_AGAS_SERVER_PORT:<hpx_initial_ip_port>}
service_mode = hosted
dedicated_server = 0
max_pending_refcnt_requests = ${HPX_AGAS_MAX_PENDING_REFCNT_REQUESTS:<hpx_initial_agas_max_pending_refcnt_requests>}
use_caching = ${HPX_AGAS_USE_CACHING:1}
use_range_caching = ${HPX_AGAS_USE_RANGE_CACHING:1}
local_cache_size = ${HPX_AGAS_LOCAL_CACHE_SIZE:<hpx_agas_local_cache_size>}
```

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hpx.agas.address</td>
<td>This property defines the default IP address to be used for the AGAS root server. This IP address will be used as long as no other values are specified (for instance using the <code>--hpx:agas</code> command line option). The expected format is any valid IP address or domain name format which can be resolved into an IP address. The default depends on the compile time preprocessor constant <code>HPX_INITIAL_IP_ADDRESS</code> (&quot;127.0.0.1&quot;).</td>
</tr>
<tr>
<td>hpx.agas.port</td>
<td>This property defines the default IP port to be used for the AGAS root server. This IP port will be used as long as no other values are specified (for instance using the <code>--hpx:agas</code> command line option). The default depends on the compile time preprocessor constant <code>HPX_INITIAL_IP_PORT</code> (7009).</td>
</tr>
<tr>
<td>hpx.agas.service_mode</td>
<td>This property specifies what type of AGAS service is running on this locality. Currently, two modes exist. The locality that acts as the AGAS server runs in bootstrap mode. All other localities are in hosted mode.</td>
</tr>
<tr>
<td>hpx.agas.dedicated_server</td>
<td>This property specifies whether the AGAS server is exclusively running AGAS services and not hosting any application components. It is a boolean value. Set to 1 if <code>--hpx:run-agas-server-only</code> is present.</td>
</tr>
<tr>
<td>hpx.agas.max_pending_refcnt_requests</td>
<td>This property defines the number of reference counting requests (increments or decrements) to buffer. The default depends on the compile time preprocessor constant <code>HPX_INITIAL_AGAS_MAX_PENDING_REFCNT_REQUESTS</code> (4096).</td>
</tr>
<tr>
<td>hpx.agas.use_caching</td>
<td>This property specifies whether a software address translation cache is used. It is a boolean value. Defaults to 1.</td>
</tr>
<tr>
<td>hpx.agas.use_range_caching</td>
<td>This property specifies whether range-based caching is used by the software address translation cache. This property is ignored if <code>hpx.agas.use_caching</code> is false. It is a boolean value. Defaults to true.</td>
</tr>
<tr>
<td>hpx.agas.local_cache_size</td>
<td>This property defines the size of the software address translation cache for AGAS services. This property is ignored if <code>hpx.agas.use_caching</code> is false. Note that if <code>hpx.agas.use_range_caching</code> is true, this size will refer to the maximum number of ranges stored in the cache, not the number of entries spanned by the cache. The default depends on the compile time preprocessor constant <code>HPX_AGAS_LOCAL_CACHE_SIZE</code> (4096).</td>
</tr>
</tbody>
</table>

The **hpx.commandline** configuration section

The following table lists the definition of all pre-defined command line option shortcuts. For more information about commandline options see the section *HPX Command Line Options.*

```plaintext
[hpX.commandline]
aliasing = ${HPX_COMMANDLINE_ALIASING:1}
allow_unknown = ${HPX_COMMANDLINE_ALLOW_UNKNOWN:0}
```

(continues on next page)
[hpx.commandlinealiases]
-a = --hpx:agas
-c = --hpx:console
-h = --hpx:help
-I = --hpx:ini
-l = --hpx:localities
-p = --hpx:app-config
-q = --hpx:queuing
-r = --hpx:runagas-server
-t = --hpx:threads
-v = --hpx:version
-w = --hpx:worker
-x = --hpx:hpx
-0 = --hpx:node=0
-1 = --hpx:node=1
-2 = --hpx:node=2
-3 = --hpx:node=3
-4 = --hpx:node=4
-5 = --hpx:node=5
-6 = --hpx:node=6
-7 = --hpx:node=7
-8 = --hpx:node=8
-9 = --hpx:node=9
<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>hpx.commandline.aliases</code></td>
<td><strong>aliasing</strong> Enable command line aliases as defined in the section <code>hpx.commandline.aliases</code> (see below). Defaults to 1.</td>
</tr>
<tr>
<td><code>hpx.commandline.aliases</code></td>
<td><strong>allow_unknown</strong> Allow for unknown command line options to be passed through to <code>hpx_main()</code> Defaults to 0.</td>
</tr>
<tr>
<td><code>hpx.commandline.aliases</code></td>
<td><strong>-a</strong> On the commandline,<code>-a</code> expands to:<code>--hpx:agas</code>.</td>
</tr>
<tr>
<td><code>hpx.commandline.aliases</code></td>
<td><strong>-c</strong> On the commandline,<code>-c</code> expands to:<code>--hpx:console</code>.</td>
</tr>
<tr>
<td><code>hpx.commandline.aliases</code></td>
<td><strong>-h</strong> On the commandline,<code>-h</code> expands to:<code>--hpx:help</code>.</td>
</tr>
<tr>
<td><code>hpx.commandline.aliases</code></td>
<td><strong>--help</strong> On the commandline,<code>--help</code> expands to:<code>--hpx:help</code>.</td>
</tr>
<tr>
<td><code>hpx.commandline.aliases</code></td>
<td><strong>-I</strong> On the commandline,<code>-I</code> expands to:<code>--hpx:ini</code>.</td>
</tr>
<tr>
<td><code>hpx.commandline.aliases</code></td>
<td><strong>-l</strong> On the commandline,<code>-l</code> expands to:<code>--hpx:localities</code>.</td>
</tr>
<tr>
<td><code>hpx.commandline.aliases</code></td>
<td><strong>-p</strong> On the commandline,<code>-p</code> expands to:<code>--hpx:app-config</code>.</td>
</tr>
<tr>
<td><code>hpx.commandline.aliases</code></td>
<td><strong>-q</strong> On the commandline,<code>-q</code> expands to:<code>--hpx:queuing</code>.</td>
</tr>
<tr>
<td><code>hpx.commandline.aliases</code></td>
<td><strong>-r</strong> On the commandline,<code>-r</code> expands to:<code>--hpx:run-agas-server</code>.</td>
</tr>
<tr>
<td><code>hpx.commandline.aliases</code></td>
<td><strong>-t</strong> On the commandline,<code>-t</code> expands to:<code>--hpx:threads</code>.</td>
</tr>
<tr>
<td><code>hpx.commandline.aliases</code></td>
<td><strong>-v</strong> On the commandline,<code>-v</code> expands to:<code>--hpx:version</code>.</td>
</tr>
<tr>
<td><code>hpx.commandline.aliases</code></td>
<td><strong>--version</strong> On the commandline,<code>--version</code> expands to:<code>--hpx:version</code>.</td>
</tr>
<tr>
<td><code>hpx.commandline.aliases</code></td>
<td><strong>-w</strong> On the commandline,<code>-w</code> expands to:<code>--hpx:worker</code>.</td>
</tr>
<tr>
<td><code>hpx.commandline.aliases</code></td>
<td><strong>-x</strong> On the commandline,<code>-x</code> expands to:<code>--hpx:hp</code>.</td>
</tr>
<tr>
<td><code>hpx.commandline.aliases</code></td>
<td><strong>-0</strong> On the commandline,<code>-0</code> expands to:<code>--hpx:node=0</code>.</td>
</tr>
<tr>
<td><code>hpx.commandline.aliases</code></td>
<td><strong>-1</strong> On the commandline,<code>-1</code> expands to:<code>--hpx:node=1</code>.</td>
</tr>
<tr>
<td><code>hpx.commandline.aliases</code></td>
<td><strong>-2</strong> On the commandline,<code>-2</code> expands to:<code>--hpx:node=2</code>.</td>
</tr>
<tr>
<td><code>hpx.commandline.aliases</code></td>
<td><strong>-3</strong> On the commandline,<code>-3</code> expands to:<code>--hpx:node=3</code>.</td>
</tr>
<tr>
<td><code>hpx.commandline.aliases</code></td>
<td><strong>-4</strong> On the commandline,<code>-4</code> expands to:<code>--hpx:node=4</code>.</td>
</tr>
<tr>
<td><code>hpx.commandline.aliases</code></td>
<td><strong>-5</strong> On the commandline,<code>-5</code> expands to:<code>--hpx:node=5</code>.</td>
</tr>
<tr>
<td><code>hpx.commandline.aliases</code></td>
<td><strong>-6</strong> On the commandline,<code>-6</code> expands to:<code>--hpx:node=6</code>.</td>
</tr>
<tr>
<td><code>hpx.commandline.aliases</code></td>
<td><strong>-7</strong> On the commandline,<code>-7</code> expands to:<code>--hpx:node=7</code>.</td>
</tr>
<tr>
<td><code>hpx.commandline.aliases</code></td>
<td><strong>-8</strong> On the commandline,<code>-8</code> expands to:<code>--hpx:node=8</code>.</td>
</tr>
</tbody>
</table>
Loading INI files

During startup and after the internal database has been initialized as described in the section *Built-in Default Configuration Settings*, **HPX** will try to locate and load additional ini files to be used as a source for configuration properties. This allows for a wide spectrum of additional customization possibilities by the user and system administrators. The sequence of locations where **HPX** will try loading the ini files is well defined and documented in this section. All ini files found are merged into the internal configuration database. The merge operation itself conforms to the rules as described in the section *The HPX INI File Format*.

1. Load all component shared libraries found in the directories specified by the property `hpx.component_path` and retrieve their default configuration information (see section *Loading components* for more details). This property can refer to a list of directories separated by `:'` (Linux, Android, and MacOS) or using `';'` (Windows).

2. Load all files named `hpx.ini` in the directories referenced by the property `hpx.master_ini_path`. This property can refer to a list of directories separated by `:'` (Linux, Android, and MacOS) or using `';'` (Windows).

3. Load a file named `.hpx.ini` in the current working directory, e.g. the directory the application was invoked from.

4. Load a file referenced by the environment variable `HPX_INI`. This variable is expected to provide the full path name of the ini configuration file (if any).

5. Load a file named `/etc/hpx.ini`. This lookup is done on non-Windows systems only.

6. Load a file named `.hpx.ini` in the home directory of the current user, e.g. the directory referenced by the environment variable `HOME`.

7. Load a file named `.hpx.ini` in the directory referenced by the environment variable `PWD`.

8. Load the file specified on the command line using the option `--hpx:config`.

9. Load all properties specified on the command line using the option `--hpx:ini`. The properties will be added to the database in the same sequence as they are specified on the command line. The format for those options is for instance `--hpx:ini=hpx.default_stack_size=0x4000`. In addition to the explicit command line options, this will set the following properties as implied from other settings:
   - `hpx.parcel.address` and `hpx.parcel.port` as set by `--hpx:hpux`
   - `hpx.agas.address`, `hpx.agas.port` and `hpx.agas.service_mode` as set by `--hpx:agas`
   - `hpx.program_name` and `hpx.cmd_line` will be derived from the actual command line
   - `hpx.os_threads` and `hpx.localities` as set by `--hpx:threads` and `--hpx:localities`
   - `hpx.runtime_mode` will be derived from any explicit `--hpx:console`, `--hpx:worker`, or `--hpx:connect`, or it will be derived from other settings, such as `--hpx:node =0` which implies `--hpx:console`

10. Load files based on the pattern `*.ini` in all directories listed by the property `hpx.ini_path`. All files found during this search will be merged. The property `hpx.ini_path` can hold a list of directories separated by `:'` (on Linux or Mac) or `';'` (on Windows).

11. Load the file specified on the command line using the option `--hpx:app-config`. Note that this file will be merged as the content for a top level section `[application]`.

**Note:** Any changes made to the configuration database caused by one of the steps will influence the loading process for all subsequent steps. For instance, if one of the ini files loaded changes the property `hpx.ini_path` this will influence the directories searched in step 9 as described above.
**Important**: The *HPX* core library will verify that all configuration settings specified on the command line (using the `--hpx:ini` option) will be checked for validity. That means that the library will accept only *known* configuration settings. This is to protect the user from unintentional typos while specifying those settings. This behavior can be overwritten by appending a `!' to the configuration key, thus forcing the setting to be entered into the configuration database, for instance: `--hpx:ini=hpx.foo! = 1`

If any of the environment variables or files listed above is not found the corresponding loading step will be silently skipped.

**Loading components**

*HPX* relies on loading application specific components during the runtime of an application. Moreover, *HPX* comes with a set of preinstalled components supporting basic functionalities useful for almost every application. Any component in *HPX* is loaded from a shared library, where any of the shared libraries can contain more than one component type. During startup, *HPX* tries to locate all available components (e.g. their corresponding shared libraries) and creates an internal component registry for later use. This section describes the algorithm used by *HPX* to locate all relevant shared libraries on a system. As described, this algorithm is customizable by the configuration properties loaded from the ini files (see section *Loading INI files*).

Loading components is a two stage process. First *HPX* tries to locate all component shared libraries, loads those, and generates default configuration section in the internal configuration database for each component found. For each found component the following information is generated:

```plaintext
[hpx.components.<component_instance_name>]
name = <name_of_shared_library>
path = ${component_path}
enabled = ${hpx.components.load_external}
default = 1
```

The values in this section correspond to the expected configuration information for a component as described in the section *Built-in Default Configuration Settings*.

In order to locate component shared libraries, *HPX* will try loading all shared libraries (files with the platform specific extension of a shared library, Linux: `*.so`, Windows: `*.dll`, MacOS: `*.dylib` found in the directory referenced by the ini property `hpx.component_path`).

This first step corresponds to step 1) during the process of filling the internal configuration database with default information as described in section *Loading INI files*.

After all of the configuration information has been loaded, *HPX* performs the second step in terms of loading components. During this step, *HPX* scans all existing configuration sections `[hpx.component.<some_component_instance_name>]` and instantiates a special factory object for each of the successfully located and loaded components. During the application’s life time, these factory objects will be responsible to create new and discard old instances of the component they are associated with. This step is performed after step 11) of the process of filling the internal configuration database with default information as described in section *Loading INI files*.

**Application specific component example**

In this section we assume to have a simple application component which exposes one member function as a component action. The header file `app_server.hpp` declares the C++ type to be exposed as a component. This type has a member function `print_greeting()` which is exposed as an action `print_greeting_action`. We assume the source files for this example are located in a directory referenced by `$APP_ROOT`:

```cpp
// file: $APP_ROOT/app_server.hpp
#include <hpx/hpx.hpp>
(continues on next page)
```
#include <hpx/include/iostreams.hpp>

namespace app {

    // Define a simple component exposing one action 'print_greeting'
    class HPX_COMPONENT_EXPORT server : public hpx::components::component_base<server> {
        void print_greeting () {
            hpx::cout << "Hey, how are you?\n" << hpx::flush;
        }

        // Component actions need to be declared, this also defines the type 'print_greeting_action' representing the action.
        HPX_DEFINE_COMPONENT_ACTION(server, print_greeting, print_greeting_action);
    };

    // Declare boilerplate code required for each of the component actions.
    HPX_REGISTER_ACTION_DECLARATION(app::server::print_greeting_action);
}

The corresponding source file contains mainly macro invocations which define boilerplate code needed for HPX to function properly:

```cpp
#include "app_server.hpp"

// Define boilerplate required once per component module.
HPX_REGISTER_COMPONENT_MODULE();

// Define factory object associated with our component of type 'app::server'.
HPX_REGISTER_COMPONENT(app::server, app_server);

// Define boilerplate code required for each of the component actions. Use the same argument as used for HPX_REGISTER_ACTION_DECLARATION above.
HPX_REGISTER_ACTION(app::server::print_greeting_action);
```

The following gives an example of how the component can be used. We create one instance of the app::server component on the current locality and invoke the exposed action print_greeting_action using the global id of the newly created instance. Note, that no special code is required to delete the component instance after it is not needed anymore. It will be deleted automatically when its last reference goes out of scope, here at the closing brace of the block surrounding the code:

```cpp
#include <hpx/hpx_init.hpp>
#include "app_server.hpp"

int hpx_main() {
    {
        // Create an instance of the app_server component on the current locality.
        hpx::naming::id_type app_server_instance = hpx::create_component<app::server>(hpx::find_here());

        // Create an instance of the action 'print_greeting_action'.
```
In order to make sure that the application will be able to use the component `app::server`, special configuration information must be passed to HPX. The simplest way to allow HPX to ‘find’ the component is to provide special ini configuration files, which add the necessary information to the internal configuration database. The component should have a special ini file containing the information specific to the component `app_server`.

### Example ini file

```
# file: $APP_ROOT/app_server.ini
[hpx.components.app_server]
name = app_server
path = $APP_LOCATION/
```

Here `$APP_LOCATION` is the directory where the (binary) component shared library is located. HPX will attempt to load the shared library from there. The section name `hpx.components.app_server` reflects the instance name of the component (`app_server` is an arbitrary, but unique name). The property value for `hpx.components.app_server.name` should be the same as used for the second argument to the macro `HPX_REGISTER_COMPONENT` above.

Additionally a file `.hpx.ini` which could be located in the current working directory (see step 3 as described in the section *Loading INI files*) can be used to add to the ini search path for components:

```
# file: $PWD/.hpx.ini
[hpx]
ini_path = $[hpx.ini_path]:$APP_ROOT/
```

This assumes that the above ini file specific to the component is located in the directory `$APP_ROOT`.

**Note:** It is possible to reference the defined property from inside its value. HPX will gracefully use the previous value of `hpx.ini_path` for the reference on the right hand side and assign the overall (now expanded) value to the property.

### Logging

HPX uses a sophisticated logging framework allowing to follow in detail what operations have been performed inside the HPX library in what sequence. This information proves to be very useful for diagnosing problems or just for improving the understanding what is happening in HPX as a consequence of invoking HPX API functionality.

#### Default logging

Enabling default logging is a simple process. The detailed description in the remainder of this section explains different ways to customize the defaults. Default logging can be enabled by using one of the following:

- a command line switch `--hpx:debug-hpx-log`, which will enable logging to the console terminal
• the command line switch \texttt{--hpx:debug-hpx-log=<filename>}, which enables logging to a given file \texttt{<filename>}, or

• setting an environment variable \texttt{HPX_LOGLEVEL=<loglevel>} while running the \texttt{HPX} application. In this case \texttt{<loglevel>} should be a number between (or equal to) \texttt{1} and \texttt{5} where \texttt{1} means minimal logging and \texttt{5} causes to log all available messages. When setting the environment variable the logs will be written to a file named \texttt{hpx.<PID>.lo} in the current working directory, where \texttt{<PID>} is the process id of the console instance of the application.

\textbf{Customizing logging}

Generally, logging can be customized either using environment variable settings or using by an ini configuration file. Logging is generated in several categories, each of which can be customized independently. All customizable configuration parameters have reasonable defaults, allowing to use logging without any additional configuration effort. The following table lists the available categories.

\begin{table}[h]
\centering
\begin{tabular}{|c|c|p{9cm}|c|}
\hline
Category & Category shortcut & Information to be generated & Environment variable \\
\hline
\textbf{General} & None & Logging information generated by different subsystems of \texttt{HPX}, such as thread-manager, parcel layer, LCOs, etc. & \texttt{HPX_LOGLEVEL} \\
\textbf{AGAS} & \texttt{AGAS} & Logging output generated by the \texttt{AGAS} subsystem & \texttt{HPX_AGAS_LOGLEVEL} \\
\textbf{Application} & \texttt{APP} & Logging generated by applications. & \texttt{HPX_APP_LOGLEVEL} \\
\hline
\end{tabular}
\caption{Logging categories}
\end{table}

By default, all logging output is redirected to the console instance of an application, where it is collected and written to a file, one file for each logging category.

Each logging category can be customized at two levels, the parameters for each are stored in the ini configuration sections \texttt{hpx.logging.CATEGORY} and \texttt{hpx.logging.console.CATEGORY} (where CATEGORY is the category shortcut as listed in the table above). The former influences logging at the source locality and the latter modifies the logging behaviour for each of the categories at the console instance of an application.

\textbf{Levels}

All \texttt{HPX} logging output has seven different logging levels. These levels can be set explicitly or through environmental variables in the main \texttt{HPX} ini file as shown below. The logging levels and their associated integral values are shown in the table below, ordered from most verbose to least verbose. By default, all \texttt{HPX} logs are set to \texttt{0}, e.g. all logging output is disabled by default.

\begin{table}[h]
\centering
\begin{tabular}{|c|c|}
\hline
Logging level & Integral value \\
\hline
\texttt{<debug>} & \texttt{5} \\
\texttt{<info>} & \texttt{4} \\
\texttt{<warning>} & \texttt{3} \\
\texttt{<error>} & \texttt{2} \\
\texttt{<fatal>} & \texttt{1} \\
No logging & \texttt{0} \\
\hline
\end{tabular}
\caption{Logging levels}
\end{table}

\textbf{Tip:} The easiest way to enable logging output is to set the environment variable corresponding to the logging category to an integral value as described in the table above. For instance, setting \texttt{HPX_LOGLEVEL=5} will enable full logging.
output for the general category. Please note that the syntax and means of setting environment variables varies between operating systems.

Configuration

Logs will be saved to destinations as configured by the user. By default, logging output is saved on the console instance of an application to hpx.<CATEGORY>.<PID>.lo (where CATEGORY and PID are placeholders for the category shortcut and the OS process id). The output for the general logging category is saved to hpx.<PID>.log. The default settings for the general logging category are shown here (the syntax is described in the section The HPX INI File Format):

```
[hpx.logging]
level = ${HPX_LOGLEVEL:0}
destination = ${HPX_LOGDESTINATION:console}
format = ${HPX_LOGFORMAT:(T%locality%/%hpxthread%/%hpxphase%/%hpxcomponent%) P
˓→%parentloc%/%hpxparent%.%hpxparentphase% %time%($hh:$mm:$ss:$mili) [%idx%]|\n}
```

The logging level is taken from the environment variable HPX_LOGLEVEL and defaults to zero, e.g. no logging. The default logging destination is read from the environment variable HPX_LOGDESTINATION. On any of the localities it defaults to console which redirects all generated logging output to the console instance of an application. The following table lists the possible destinations for any logging output. It is possible to specify more than one destination separated by whitespace.

<table>
<thead>
<tr>
<th>Logging destination</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file(&lt;filename&gt;)</td>
<td>Direct all output to a file with the given &lt;filename&gt;.</td>
</tr>
<tr>
<td>cout</td>
<td>Direct all output to the local standard output of the application instance on this locality.</td>
</tr>
<tr>
<td>cerr</td>
<td>Direct all output to the local standard error output of the application instance on this locality.</td>
</tr>
<tr>
<td>console</td>
<td>Direct all output to the console instance of the application. The console instance has its logging destinations configured separately.</td>
</tr>
<tr>
<td>android_log</td>
<td>Direct all output to the (Android) system log (available on Android systems only).</td>
</tr>
</tbody>
</table>

The logging format is read from the environment variable HPX_LOGFORMAT and it defaults to a complex format description. This format consists of several placeholder fields (for instance %locality% which will be replaced by concrete values when the logging output is generated. All other information is transferred verbatim to the output. The table below describes the available field placeholders. The separator character | separates the logging message prefix formatted as shown and the actual log message which will replace the separator.
### Table 2.10: Available field placeholders

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>locality</td>
<td>The id of the <em>locality</em> on which the logging message was generated.</td>
</tr>
<tr>
<td>hpxthread</td>
<td>The id of the HPX-thread generating this logging output.</td>
</tr>
<tr>
<td>hpxphase</td>
<td>The phase(^{111}) of the HPX-thread generating this logging output.</td>
</tr>
<tr>
<td>hpxcomponent</td>
<td>The local virtual address of the component which the current HPX-thread is accessing.</td>
</tr>
<tr>
<td>parentloc</td>
<td>The id of the <em>locality</em> where the HPX thread was running which initiated the current HPX-thread. The current HPX-thread is generating this logging output.</td>
</tr>
<tr>
<td>hpxparent</td>
<td>The id of the HPX-thread which initiated the current HPX-thread. The current HPX-thread is generating this logging output.</td>
</tr>
<tr>
<td>hpxparentphase</td>
<td>The phase of the HPX-thread when it initiated the current HPX-thread. The current HPX-thread is generating this logging output.</td>
</tr>
<tr>
<td>time</td>
<td>The time stamp for this logging output line as generated by the source <em>locality</em>.</td>
</tr>
<tr>
<td>idx</td>
<td>The sequence number of the logging output line as generated on the source <em>locality</em>.</td>
</tr>
<tr>
<td>osthread</td>
<td>The sequence number of the OS-thread which executes the current HPX-thread.</td>
</tr>
</tbody>
</table>

**Note:** Not all of the field placeholder may be expanded for all generated logging output. If no value is available for a particular field it is replaced with a sequence of "-" characters.

Here is an example line from a logging output generated by one of the HPX examples (please note that this is generated on a single line, without line break):

```
(T00000000/0000000002d46f90.01/00000000009ebc10) P--------/0000000002d46f80.02 17:49.
˓
→ 37.320 [000000000000004d] <info> [RT] successfully created component {0000000100ff0001, 000000000030002}, of type: component_barrier[7(3)]
```

The default settings for the general logging category on the console is shown here:

```
[hpx.logging.console]
level = ${HPX_LOGLEVEL:$[hpx.logging.level]}
destination = ${HPX_CONSOLE_LOGDESTINATION:file(hpx.$[system.pid].log)}
format = ${HPX_CONSOLE_LOGFORMAT:}
```

These settings define how the logging is customized once the logging output is received by the console instance of an application. The logging level is read from the environment variable `HPX_LOGLEVEL` (as set for the console instance of the application). The level defaults to the same values as the corresponding settings in the general logging configuration shown before. The destination on the console instance is set to be a file which name is generated based from its OS process id. Setting the environment variable `HPX_CONSOLE_LOGDESTINATION` allows customization of the naming scheme for the output file. The logging format is set to leave the original logging output unchanged, as received from one of the localities the application runs on.

**HPX Command Line Options**

The predefined command line options for any application using `hpx::init` are described in the following subsections.

\(^{111}\) The phase of a HPX-thread counts how often this thread has been activated.
**HPX options (allowed on command line only)**

```bash
--hpx:help
  print out program usage (default: this message), possible values: full (additionally prints options from components)
```

```bash
--hpx:version
  print out HPX version and copyright information
```

```bash
--hpx:info
  print out HPX configuration information
```

```bash
--hpx:options-file arg
  specify a file containing command line options (alternatively: @filepath)
```

**HPX options (additionally allowed in an options file)**

```bash
--hpx:worker
  run this instance in worker mode
```

```bash
--hpx:console
  run this instance in console mode
```

```bash
--hpx:connect
  run this instance in worker mode, but connecting late
```

```bash
--hpx:run-agas-server
  run AGAS server as part of this runtime instance
```

```bash
--hpx:run-hpx-main
  run the hpx_main function, regardless of locality mode
```

```bash
--hpx:hp arg
  the IP address the HPX parcelport is listening on, expected format: address:port (default: 127.0.0.1:7910)
```

```bash
--hpx:agas arg
  the IP address the AGAS root server is running on, expected format: address:port (default: 127.0.0.1:7910)
```

```bash
--hpx:run-agas-server-only
  run only the AGAS server
```

```bash
--hpx:nodefile arg
  the file name of a node file to use (list of nodes, one node name per line and core)
```

```bash
--hpx:nodes arg
  the (space separated) list of the nodes to use (usually this is extracted from a node file)
```

```bash
--hpx:endnodes
  this can be used to end the list of nodes specified using the option --hpx:nodes
```

```bash
--hpx:ifsuffix arg
  suffix to append to host names in order to resolve them to the proper network interconnect
```

```bash
--hpx:ifprefix arg
  prefix to prepend to host names in order to resolve them to the proper network interconnect
```

```bash
--hpx:iftransform arg
  sed-style search and replace (s/search/replace/) used to transform host names to the proper network interconnect
```
--hpx:localities arg
    the number of localities to wait for at application startup (default: 1)

--hpx:node arg
    number of the node this locality is run on (must be unique)

--hpx:ignore-batch-env
    ignore batch environment variables

--hpx:expect-connecting-localities
    this locality expects other localities to dynamically connect (this is implied if the number of initial localities is larger than 1)

--hpx:pu-offset
    the first processing unit this instance of HPX should be run on (default: 0)

--hpx:pu-step
    the step between used processing unit numbers for this instance of HPX (default: 1)

--hpx:threads arg
    the number of operating system threads to spawn for this HPX locality. Possible values are: numeric values 1, 2, 3 and so on, all (which spawns one thread per processing unit, includes hyperthreads), or cores (which spawns one thread per core) (default: cores).

--hpx:cores arg
    the number of cores to utilize for this HPX locality (default: all, i.e. the number of cores is based on the number of threads --hpx:threads assuming --hpx:bind=compact

--hpx:affinity arg
    the affinity domain the OS threads will be confined to, possible values: pu, core, numa, machine (default: pu)

--hpx:bind arg
    the detailed affinity description for the OS threads, see More details about HPX command line options for a detailed description of possible values. Do not use with --hpx:pu-step, --hpx:pu-offset or --hpx:affinity options. Implies --hpx:numa-sensitive (--hpx:bind=none) disables defining thread affinities.

--hpx:use-process-mask
    use the process mask to restrict available hardware resources (implies --hpx:ignore-batch-env)

--hpx:print-bind
    print to the console the bit masks calculated from the arguments specified to all --hpx:bind options.

--hpx:queuing arg
    the queue scheduling policy to use, options are local, local-priority-fifo, local-priority-lifo, static, static-priority, abp-priority-fifo and abp-priority-lifo (default: local-priority-fifo)

--hpx:high-priority-threads arg
    the number of operating system threads maintaining a high priority queue (default: number of OS threads), valid for --hpx:queuing=abp-priority, --hpx:queuing=static-priority and --hpx:queuing=local-priority only

--hpx:numa-sensitive
    makes the scheduler NUMA sensitive

**HPX configuration options**

--hpx:app-config arg
    load the specified application configuration (ini) file
--hpx:config arg
load the specified hpx configuration (ini) file

--hpx:ini arg
add a configuration definition to the default runtime configuration

--hpx:exit
exit after configuring the runtime

**HPX debugging options**

--hpx:list-symbolic-names
list all registered symbolic names after startup

--hpx:list-component-types
list all dynamic component types after startup

--hpx:dump-config-initial
print the initial runtime configuration

--hpx:dump-config
print the final runtime configuration

--hpx:debug-hpx-log [arg]
enable all messages on the HPX log channel and send all HPX logs to the target destination (default: cout)

--hpx:debug-agas-log [arg]
enable all messages on the AGAS log channel and send all AGAS logs to the target destination (default: cout)

--hpx:debug-parcel-log [arg]
enable all messages on the parcel transport log channel and send all parcel transport logs to the target destination (default: cout)

--hpx:debug-timing-log [arg]
enable all messages on the timing log channel and send all timing logs to the target destination (default: cout)

--hpx:debug-app-log [arg]
enable all messages on the application log channel and send all application logs to the target destination (default: cout)

--hpx:debug-clp
debag command line processing

--hpx:attach-debugger arg
wait for a debugger to be attached, possible arg values: startup or exception (default: startup)

**HPX options related to performance counters**

--hpx:print-counter
print the specified performance counter either repeatedly and/or at the times specified by --hpx:print-counter-at (see also option --hpx:print-counter-interval)

--hpx:print-counter-reset
print the specified performance counter either repeatedly and/or at the times specified by --hpx:print-counter-at reset the counter after the value is queried. (see also option --hpx:print-counter-interval)

--hpx:print-counter-interval
print the performance counter(s) specified with --hpx:print-counter repeatedly after the time interval (specified in milliseconds), (default: 0, which means print once at shutdown)
--hpx:print-counter-destination
   print the performance counter(s) specified with --hpx:print-counter to the given file (default: console)

--hpx:list-counters
   list the names of all registered performance counters, possible values: minimal (prints counter name skeletons), full (prints all available counter names)

--hpx:list-counter-infos
   list the description of all registered performance counters, possible values: minimal (prints info for counter name skeletons), full (prints all available counter infos)

--hpx:print-counter-format
   print the performance counter(s) specified with --hpx:print-counter possible formats in csv format with header or without any header (see option --hpx:no-csv-header, possible values: csv (prints counter values in CSV format with full names as header), csv-short (prints counter values in CSV format with shortnames provided with --hpx:print-counter as --hpx:print-counter shortname), full-countername

--hpx:no-csv-header
   print the performance counter(s) specified with --hpx:print-counter and csv or csv-short format specified with --hpx:print-counter-format without header

--hpx:print-counter-at arg
   print the performance counter(s) specified with --hpx:print-counter (or --hpx:print-counter-reset at the given point in time, possible argument values: startup, shutdown (default), noshutdown

--hpx:reset-counters
   reset all performance counter(s) specified with --hpx:print-counter after they have been evaluated.

--hpx:print-counters-locally
   Each locality prints only its own local counters. If this is used with --hpx:print-counter-destination=<file>, the code will append a ".<locality_id>" to the file name in order to avoid clashes between localities.

Command line argument shortcuts

Additionally, the following shortcuts are available from every HPX application.
Table 2.11: Predefined command line option shortcuts

<table>
<thead>
<tr>
<th>Shortcut option</th>
<th>Equivalent long option</th>
</tr>
</thead>
<tbody>
<tr>
<td>-a</td>
<td>--hpx:agas</td>
</tr>
<tr>
<td>-c</td>
<td>--hpx:console</td>
</tr>
<tr>
<td>-h</td>
<td>--hpx:help</td>
</tr>
<tr>
<td>-I</td>
<td>--hpx:ini</td>
</tr>
<tr>
<td>-l</td>
<td>--hpx:localities</td>
</tr>
<tr>
<td>-p</td>
<td>--hpx:app-config</td>
</tr>
<tr>
<td>-q</td>
<td>--hpx:queuing</td>
</tr>
<tr>
<td>-r</td>
<td>--hpx:run-agas-server</td>
</tr>
<tr>
<td>-t</td>
<td>--hpx:threads</td>
</tr>
<tr>
<td>-v</td>
<td>--hpx:version</td>
</tr>
<tr>
<td>-w</td>
<td>--hpx:worker</td>
</tr>
<tr>
<td>-x</td>
<td>--hpx:hpx</td>
</tr>
<tr>
<td>-0</td>
<td>--hpx:node=0</td>
</tr>
<tr>
<td>-1</td>
<td>--hpx:node=1</td>
</tr>
<tr>
<td>-2</td>
<td>--hpx:node=2</td>
</tr>
<tr>
<td>-3</td>
<td>--hpx:node=3</td>
</tr>
<tr>
<td>-4</td>
<td>--hpx:node=4</td>
</tr>
<tr>
<td>-5</td>
<td>--hpx:node=5</td>
</tr>
<tr>
<td>-6</td>
<td>--hpx:node=6</td>
</tr>
<tr>
<td>-7</td>
<td>--hpx:node=7</td>
</tr>
<tr>
<td>-8</td>
<td>--hpx:node=8</td>
</tr>
<tr>
<td>-9</td>
<td>--hpx:node=9</td>
</tr>
</tbody>
</table>

It is possible to define your own shortcut options. In fact, all of the shortcuts listed above are pre-defined using the technique described here. Also, it is possible to redefine any of the pre-defined shortcuts to expand differently as well.

Shortcut options are obtained from the internal configuration database. They are stored as key-value properties in a special properties section named `hpx.commandline`. You can define your own shortcuts by adding the corresponding definitions to one of the ini configuration files as described in the section *Configuring HPX applications*. For instance, in order to define a command line shortcut `--p` which should expand to `--hpx:print-counter`, the following configuration information needs to be added to one of the ini configuration files:

```
[hpx.commandline.aliases]
--pc = --hpx:print-counter
```

**Note:** Any arguments for shortcut options passed on the command line are retained and passed as arguments to the corresponding expanded option. For instance, given the definition above, the command line option:

```
--pc=/threads{locality#0/total}/count/cumulative
```

would be expanded to:

```
--hpx:print-counter=/threads{locality#0/total}/count/cumulative
```

**Important:** Any shortcut option should either start with a single '-' or with two '---' characters. Shortcuts starting with a single '-' are interpreted as short options (i.e. everything after the first character following the '-' is treated as the argument). Shortcuts starting with '---' are interpreted as long options. No other shortcut formats are supported.

2.5. Manual 129
Specifying options for single localities only

For runs involving more than one locality it is sometimes desirable to supply specific command line options to single localities only. When the HPX application is launched using a scheduler (like PBS, for more details see section How to use HPX applications with PBS), specifying dedicated command line options for single localities may be desirable. For this reason all of the command line options which have the general format `--hpx:<some_key>` can be used in a more general form: `--hpx:<N>:<some_key>`, where `<N>` is the number of the locality this command line options will be applied to, all other localities will simply ignore the option. For instance, the following PBS script passes the option `--hpx:pu-offset=4` to the locality '1' only.

```bash
#!/bin/bash
#PBS -l nodes=2:ppn=4
APP_PATH=~/packages/hpx/bin/hello_world_distributed
APP_OPTIONS=

pbsdsh -u $APP_PATH $APP_OPTIONS --hpx:1:pu-offset=4 --hpx:pu-offset=4 --hpx:nodes=`cat $PBS_NODEFILE`
```

Caution: If the first application specific argument (inside `$APP_OPTIONS` is a non-option (i.e. does not start with a `-` or a `--`), then it must be placed before the option `--hpx:nodes`, which, in this case, should be the last option on the command line.

Alternatively, use the option `--hpx:endnodes` to explicitly mark the end of the list of node names:

```bash
pbsdsh -u $APP_PATH $APP_OPTIONS --hpx:1:pu-offset=4 --hpx:nodes=`cat $PBS_NODEFILE` --
  hpx:endnodes $APP_OPTIONS
```

More details about HPX command line options

This section documents the following list of the command line options in more detail:

- The command line option `--hpx:bind`

The command line option `--hpx:bind`

This command line option allows one to specify the required affinity of the HPX worker threads to the underlying processing units. As a result the worker threads will run only on the processing units identified by the corresponding bind specification. The affinity settings are to be specified using `--hpx:bind=<BINDINGS>`, where `<BINDINGS>` have to be formatted as described below.

In addition to the syntax described below one can use `--hpx:bind=None` to disable all binding of any threads to a particular core. This is mostly supported for debugging purposes.

The specified affinities refer to specific regions within a machine hardware topology. In order to understand the hardware topology of a particular machine it may be useful to run the lstopo tool which is part of Portable Hardware Locality (HWLOC) to see the reported topology tree. Seeing and understanding a topology tree will definitely help in understanding the concepts that are discussed below.

Affinities can be specified using HWLOC (Portable Hardware Locality (HWLOC)) tuples. Tuples of HWLOC objects and associated indexes can be specified in the form `object:index, object:index-index or object:index,...,index`. HWLOC objects represent types of mapped items in a topology tree. Possible values for objects are `socket`, `numanode`, `core` and `pu` (processing unit). Indexes are non-negative integers that specify a unique physical object in a topology tree using its logical sequence number.

Chaining multiple tuples together in the more general form `object1:index1[, object2:index2[...]]` is
permissible. While the first tuple’s object may appear anywhere in the topology, the Nth tuple’s object must have a shallower topology depth than the (N+1)th tuple’s object. Put simply: as you move right in a tuple chain, objects must go deeper in the topology tree. Indexes specified in chained tuples are relative to the scope of the parent object. For example, socket:0.core:1 refers to the second core in the first socket (all indices are zero based).

Multiple affinities can be specified using several --hpx:bind command line options or by appending several affinities separated by a ';'. By default, if multiple affinities are specified, they are added.

"all" is a special affinity consisting in the entire current topology.

**Note:** All ‘names’ in an affinity specification, such as thread, socket, numanode, pu or all can be abbreviated. Thus the affinity specification threads:0-3=socket:0.core:1.pu:1 is fully equivalent to its shortened form t:0-3=s:0.c:1.p:1.

Here is a full grammar describing the possible format of mappings:

```
mappings ::= distribution | mapping (";" mapping)*
distribution ::= "compact" | "scatter" | "balanced" | "numa-balanced"
mapping ::= thread_spec "=" pu_specs
thread_spec ::= "thread:" range_specs
pu_specs ::= pu_spec ("." pu_spec)*
pu_spec ::= type ":" range_specs | ":" pu_spec
range_specs ::= range_spec ("," range_spec)*
range_spec ::= int | int "-" int | "all"
type ::= "socket" | "numanode" | "core" | "pu"
```

The following example assumes a system with at least 4 cores, where each core has more than 1 processing unit (hardware threads). Running hello_world_distributed with 4 OS-threads (on 4 processing units), where each of those threads is bound to the first processing unit of each of the cores, can be achieved by invoking:

```
hello_world_distributed -t4 --hpx:bind=thread:0-3=core:0-3.pu:0
```

Here thread:0-3 specifies the OS threads for which to define affinity bindings, and core:0-3.pu:0 defines that for each of the cores (core:0-3) only their first processing unit pu:0 should be used.

**Note:** The command line option --hpx:print-bind can be used to print the bitmasks generated from the affinity mappings as specified with --hpx:bind. For instance, on a system with hyperthreading enabled (i.e. 2 processing units per core), the command line:

```
hello_world_distributed -t4 --hpx:bind=thread:0-3=core:0-3.pu:0 --hpx:print-bind
```

will cause this output to be printed:

```
0: PU L#0(P#0), Core L#0, Socket L#0, Node L#0(P#0)
1: PU L#2(P#2), Core L#1, Socket L#0, Node L#0(P#0)
2: PU L#4(P#4), Core L#2, Socket L#0, Node L#0(P#0)
3: PU L#6(P#6), Core L#3, Socket L#0, Node L#0(P#0)
```

where each bit in the bitmasks corresponds to a processing unit the listed worker thread will be bound to run on.

The difference between the four possible predefined distribution schemes (compact, scatter, balanced and numa-balanced) is best explained with an example. Imagine that we have a system with 4 cores and 4 hardware threads per core on 2 sockets. If we place 8 threads the assignments produced by the compact, scatter,
balanced and numa-balanced types are shown in the figure below. Notice that compact does not fully utilize all the cores in the system. For this reason it is recommended that applications are run using the scatter or balanced/numa-balanced options in most cases.

In addition to the predefined distributions it is possible to restrict the resources used by HPX to the process CPU mask. The CPU mask is typically set by e.g. MPI\(^\text{110}\) and batch environments. Using the command line option --hpx:use-process-mask makes HPX act as if only the processing units in the CPU mask are available for use by HPX. The number of threads is automatically determined from the CPU mask. The number of threads can still be changed manually using this option, but only to a number less than or equal to the number of processing units in the CPU mask. The option --hpx:print-bind is useful in conjunction with --hpx:use-process-mask to make sure threads are placed as expected.

### 2.5.6 Writing single-node HPX applications

HPX is a C++ Standard Library for Concurrency and Parallelism. This means that it implements all of the corresponding facilities as defined by the C++ Standard. Additionally, HPX implements functionalities proposed as part

\(^{110}\) https://en.wikipedia.org/wiki/Message_Passing_Interface
of the ongoing C++ standardization process. This section focuses on the features available in HPX for parallel and concurrent computation on a single node, although many of the features presented here are also implemented to work in the distributed case.

Using LCOs

Lightweight Control Objects (LCOs) provide synchronization for HPX applications. Most of them are familiar from other frameworks, but a few of them work in slightly different ways adapted to HPX. The following synchronization objects are available in HPX:

1. future
2. queue
3. object_semaphore
4. barrier

Channels

Channels combine communication (the exchange of a value) with synchronization (guaranteeing that two calculations (tasks) are in a known state). A channel can transport any number of values of a given type from a sender to a receiver:

```cpp
hpx::lcos::local::channel<int> c;
c.set(42);
cout << c.get(); // will print '42'
```

Channels can be handed to another thread (or in case of channel components, to other localities), thus establishing a communication channel between two independent places in the program:

```cpp
do_something(
    hpx::lcos::local::receive_channel<int> c,
    hpx::lcos::local::send_channel<> done)
{
    cout << c.get(); // prints 42
    done.set(); // signal back
}
```

```cpp
hpx::lcos::local::channel<int> c;
hpx::lcos::local::channel<> done;
hpx::apply(&do_something, c, done);
c.set(42); // send some value
done.get(); // wait for thread to be done
```

A channel component is created on one `locality` and can be sent to another `locality` using an action. This example also demonstrates how a channel can be used as a range of values:

```cpp
// channel components need to be registered for each used type (not needed
// for hpx::lcos::local::channel)
HPX_REGISTER_CHANNEL(double);

void some_action(hpx::lcos::channel<double> c)
{
    for (double d : c)
```
hpx::cout << d << std::endl;
}
HPX_REGISTER_ACTION(some_action);
{
    // create the channel on this locality
    hpx::lcos::channel<double> c(hpx::find_here());

    // pass the channel to a (possibly remote invoked) action
    hpx::apply(some_action(), hpx::find_here(), c);

    // send some values to the receiver
    std::vector<double> v = {1.2, 3.4, 5.0};
    for (double d : v)
        c.set(d);

    // explicitly close the communication channel (implicit at destruction)
    c.close();
}

Composable guards

Composable guards operate in a manner similar to locks, but are applied only to asynchronous functions. The guard (or guards) is automatically locked at the beginning of a specified task and automatically unlocked at the end. Because guards are never added to an existing task’s execution context, the calling of guards is freely composable and can never deadlock.

To call an application with a single guard, simply declare the guard and call run_guarded() with a function (task):

```cpp
hpx::lcos::local::guard gu;
run_guarded(gu, task);
```

If a single method needs to run with multiple guards, use a guard set:

```cpp
boost::shared<hpx::lcos::local::guard> gu1(new hpx::lcos::local::guard());
boost::shared<hpx::lcos::local::guard> gu2(new hpx::lcos::local::guard());
gs.add(*gu1);
gs.add(*gu2);
run_guarded(gs, task);
```

Guards use two atomic operations (which are not called repeatedly) to manage what they do, so overhead should be extremely low. The following guards are available in HPX:

1. conditional_trigger
2. counting_semaphore
3. dataflow
4. event
5. mutex
6. once
7. recursive_mutex
8. spinlock
9. spinlock_no_backoff
10. **trigger**

**Extended facilities for futures**

Concurrency is about both decomposing and composing the program from the parts that work well individually and together. It is in the composition of connected and multicore components where today’s C++ libraries are still lacking.

The functionality of `std::future` offers a partial solution. It allows for the separation of the initiation of an operation and the act of waiting for its result; however, the act of waiting is synchronous. In communication-intensive code this act of waiting can be unpredictable, inefficient and simply frustrating. The example below illustrates a possible synchronous wait using futures:

```cpp
#include <future>
using namespace std;

int main()
{
    future<int> f = async([](){ return 123; });
    int result = f.get(); // might block
}
```

For this reason, HPX implements a set of extensions to `std::future` (as proposed by __cpp11_n4107__). This proposal introduces the following key asynchronous operations to `hpx::future`, `hpx::shared_future` and `hpx::async`, which enhance and enrich these facilities.

<table>
<thead>
<tr>
<th>Facility</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hpx::future::then</td>
<td>In asynchronous programming, it is very common for one asynchronous operation, on completion, to invoke a second operation and pass data to it. The current C++ standard does not allow one to register a continuation to a future. With <code>then</code>, instead of waiting for the result, a continuation is “attached” to the asynchronous operation, which is invoked when the result is ready. Continuations registered using <code>then</code> function will help to avoid blocking waits or wasting threads on polling, greatly improving the responsiveness and scalability of an application.</td>
</tr>
<tr>
<td>unwrapping constructor for hpx::future</td>
<td>In some scenarios, you might want to create a future that returns another future, resulting in nested futures. Although it is possible to write code to unwrap the outer future and retrieve the nested future and its result, such code is not easy to write because users must handle exceptions and it may cause a blocking call. Unwrapping can allow users to mitigate this problem by doing an asynchronous call to unwrap the outermost future.</td>
</tr>
<tr>
<td>hpx::future::is_ready</td>
<td>There are often situations where a <code>get()</code> call on a future may not be a blocking call, or is only a blocking call under certain circumstances. This function gives the ability to test for early completion and allows us to avoid associating a continuation, which needs to be scheduled with some non-trivial overhead and near-certain loss of cache efficiency.</td>
</tr>
<tr>
<td>hpx::make_ready_future</td>
<td>Some functions may know the value at the point of construction. In these cases the value is immediately available, but needs to be returned as a future. By using <code>hpx::make_ready_future</code> a future can be created that holds a pre-computed result in its shared state. In the current standard it is non-trivial to create a future directly from a value. First a promise must be created, then the promise is set, and lastly the future is retrieved from the promise. This can now be done with one operation.</td>
</tr>
</tbody>
</table>

The standard also omits the ability to compose multiple futures. This is a common pattern that is ubiquitous in other asynchronous frameworks and is absolutely necessary in order to make C++ a powerful asynchronous programming language. Not including these functions is synonymous to Boolean algebra without AND/OR.

In addition to the extensions proposed by N4313[^112], HPX adds functions allowing users to compose several futures in

a more flexible way.

Table 2.14: Facilities for composing `hpx::future`

<table>
<thead>
<tr>
<th>Facility</th>
<th>Description</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>hpx::when_any,</code></td>
<td>Asynchronously wait for at least one of multiple future</td>
<td><code>N4313\[^{113}\], \ldots\_n versions are HPX only</code></td>
</tr>
<tr>
<td><code>hpx::when_any_n</code></td>
<td>or shared_future objects to finish.</td>
<td></td>
</tr>
<tr>
<td><code>hpx::wait_any,</code></td>
<td>Synchronously wait for at least one of multiple future</td>
<td><code>HPX only</code></td>
</tr>
<tr>
<td><code>hpx::wait_any_n</code></td>
<td>or shared_future objects to finish.</td>
<td></td>
</tr>
<tr>
<td><code>hpx::when_all,</code></td>
<td>Asynchronously wait for all future and shared_future</td>
<td><code>N4313\[^{114}\], \ldots\_n versions are HPX only</code></td>
</tr>
<tr>
<td><code>hpx::when_all_n</code></td>
<td>objects to finish.</td>
<td></td>
</tr>
<tr>
<td><code>hpx::wait_all,</code></td>
<td>Synchronously wait for all future and shared_future</td>
<td><code>HPX only</code></td>
</tr>
<tr>
<td><code>hpx::wait_all_n</code></td>
<td>objects to finish.</td>
<td></td>
</tr>
<tr>
<td><code>hpx::when_some,</code></td>
<td>Asynchronously wait for multiple future and shared_future</td>
<td><code>HPX only</code></td>
</tr>
<tr>
<td><code>hpx::when_some_n</code></td>
<td>objects to finish.</td>
<td></td>
</tr>
<tr>
<td><code>hpx::wait_some,</code></td>
<td>Synchronously wait for multiple future and shared_future</td>
<td><code>HPX only</code></td>
</tr>
<tr>
<td><code>hpx::wait_some_n</code></td>
<td>objects to finish.</td>
<td></td>
</tr>
<tr>
<td><code>hpx::when_each</code></td>
<td>Asynchronously wait for multiple future and shared_future</td>
<td><code>HPX only</code></td>
</tr>
<tr>
<td><code>hpx::wait_each,</code></td>
<td>objects to finish and call a function for each of the</td>
<td></td>
</tr>
<tr>
<td><code>hpx::wait_each_n</code></td>
<td>future objects as soon as it becomes ready.</td>
<td></td>
</tr>
</tbody>
</table>

High level parallel facilities

In preparation for the upcoming C++ Standards, there are currently several proposals targeting different facilities supporting parallel programming. `HPX` implements (and extends) some of those proposals. This is well aligned with our strategy to align the APIs exposed from `HPX` with current and future C++ Standards.

At this point, `HPX` implements several of the C++ Standardization working papers, most notably `N4409\[^{115}\]` (Working Draft, Technical Specification for C++ Extensions for Parallelism), `N4411\[^{116}\]` (Task Blocks), and `N4406\[^{117}\]` (Parallel Algorithms Need Executors).

Using parallel algorithms

A parallel algorithm is a function template described by this document which is declared in the (inline) namespace `hpx::parallel::v1`.

Note: For compilers that do not support inline namespaces, all of the namespace `v1` is imported into the namespace `hpx::parallel`. The effect is similar to what inline namespaces would do, namely all names defined in `hpx::parallel::v1` are accessible from the namespace `hpx::parallel` as well.

All parallel algorithms are very similar in semantics to their sequential counterparts (as defined in the namespace `std`) with an additional formal template parameter named `ExecutionPolicy`. The execution policy is generally passed as the first argument to any of the parallel algorithms and describes the manner in which the execution of these algorithms may be parallelized and the manner in which they apply user-provided function objects.

\[^{113}\] http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2014/n4313.html
\[^{114}\] http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2014/n4313.html
\[^{115}\] http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2015/n4409.pdf
\[^{117}\] http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2015/n4406.pdf
The applications of function objects in parallel algorithms invoked with an execution policy object of type `hpx::parallel::execution::sequenced_policy` or `hpx::parallel::execution::sequenced_task_policy` execute in sequential order. For `hpx::parallel::execution::sequenced_policy` the execution happens in the calling thread.

The applications of function objects in parallel algorithms invoked with an execution policy object of type `hpx::parallel::execution::parallel_policy` or `hpx::parallel::execution::parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and are indeterminately sequenced within each thread.

**Important:** It is the caller’s responsibility to ensure correctness, such as making sure that the invocation does not introduce data races or deadlocks.

The applications of function objects in parallel algorithms invoked with an execution policy of type `hpx::parallel::execution::parallel_unsequenced_policy` is, in HPX, equivalent to the use of the execution policy `hpx::parallel::execution::parallel_policy`.

Algorithms invoked with an execution policy object of type `hpx::parallel::v1::execution_policy` execute internally as if invoked with the contained execution policy object. No exception is thrown when an `hpx::parallel::v1::execution_policy` contains an execution policy of type `hpx::parallel::execution::sequenced_policy` or `hpx::parallel::execution::parallel_task_policy` (which normally turn the algorithm into its asynchronous version). In this case the execution is semantically equivalent to the case of passing a `hpx::parallel::execution::sequenced_policy` or `hpx::parallel::execution::parallel_policy` contained in the `hpx::parallel::v1::execution_policy` object respectively.

**Parallel exceptions**

During the execution of a standard parallel algorithm, if temporary memory resources are required by any of the algorithms and no memory is available, the algorithm throws a `std::bad_alloc` exception.

During the execution of any of the parallel algorithms, if the application of a function object terminates with an uncaught exception, the behavior of the program is determined by the type of execution policy used to invoke the algorithm:

- If the execution policy object is of type `hpx::parallel::execution::parallel_unsequenced_policy`, `hpx::terminate` shall be called.
- If the execution policy object is of type `hpx::parallel::execution::sequenced_policy`, `hpx::parallel::execution::sequenced_task_policy`, `hpx::parallel::execution::parallel_policy` or `hpx::parallel::execution::parallel_task_policy`, the execution of the algorithm terminates with an `hpx::exception_list` exception. All uncaught exceptions thrown during the application of user-provided function objects shall be contained in the `hpx::exception_list` object.

For example, the number of invocations of the user-provided function object in `for_each` is unspecified. When `hpx::parallel::v1::for_each` is executed sequentially, only one exception will be contained in the `hpx::exception_list` object.

These guarantees imply that, unless the algorithm has failed to allocate memory and terminated with `std::bad_alloc`, all exceptions thrown during the execution of the algorithm are communicated to the caller. It is unspecified whether an algorithm implementation will “forge ahead” after encountering and capturing a user exception.

The algorithm may terminate with the `std::bad_alloc` exception even if one or more user-provided function objects have terminated with an exception. For example, this can happen when an algorithm fails to allocate memory while creating or adding elements to the `hpx::exception_list` object.
Parallel algorithms

*HPX* provides implementations of the following parallel algorithms:
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>In header</th>
<th>Algorithm page at cppreference.com</th>
</tr>
</thead>
<tbody>
<tr>
<td>hpx::parallel::v1::adjacent_find</td>
<td>Computes the differences between adjacent elements in a range.</td>
<td><code>&lt;hpx/include/parallel_adjacent_find.hpp&gt;</code></td>
<td><code>adjacent_find</code>&lt;sup&gt;18&lt;/sup&gt;</td>
</tr>
<tr>
<td>hpx::parallel::v1::all_of</td>
<td>Checks if a predicate is true for all of the elements in a range.</td>
<td><code>&lt;hpx/include/parallel_all_any_none.hpp&gt;</code></td>
<td><code>all_any_none_of</code>&lt;sup&gt;20&lt;/sup&gt;</td>
</tr>
<tr>
<td>hpx::parallel::v1::any_of</td>
<td>Checks if a predicate is true for any of the elements in a range.</td>
<td><code>&lt;hpx/include/parallel_all_any_none.hpp&gt;</code></td>
<td><code>all_any_none_of</code>&lt;sup&gt;20&lt;/sup&gt;</td>
</tr>
<tr>
<td>hpx::parallel::v1::count</td>
<td>Returns the number of elements equal to a given value.</td>
<td><code>&lt;hpx/include/parallel_count.hpp&gt;</code></td>
<td><code>count&lt;sup&gt;21&lt;/sup&gt;</code></td>
</tr>
<tr>
<td>hpx::parallel::v1::count_if</td>
<td>Returns the number of elements satisfying a specific criteria.</td>
<td><code>&lt;hpx/include/parallel_count.hpp&gt;</code></td>
<td><code>count_if&lt;sup&gt;22&lt;/sup&gt;</code></td>
</tr>
<tr>
<td>hpx::parallel::v1::equal</td>
<td>Determines if two sets of elements are the same.</td>
<td><code>&lt;hpx/include/parallel_equal.hpp&gt;</code></td>
<td><code>equal&lt;sup&gt;23&lt;/sup&gt;</code></td>
</tr>
<tr>
<td>hpx::parallel::v1::exclusive_scan</td>
<td>Does an exclusive parallel scan over a range of elements.</td>
<td><code>&lt;hpx/include/parallel_scan.hpp&gt;</code></td>
<td><code>exclusive_scan&lt;sup&gt;14&lt;/sup&gt;</code></td>
</tr>
<tr>
<td>hpx::parallel::v1::find</td>
<td>Finds the first element equal to a given value.</td>
<td><code>&lt;hpx/include/parallel_find.hpp&gt;</code></td>
<td><code>find&lt;sup&gt;25&lt;/sup&gt;</code></td>
</tr>
<tr>
<td>hpx::parallel::v1::find_end</td>
<td>Finds the last sequence of elements in a certain range.</td>
<td><code>&lt;hpx/include/parallel_find.hpp&gt;</code></td>
<td><code>find_end&lt;sup&gt;26&lt;/sup&gt;</code></td>
</tr>
<tr>
<td>hpx::parallel::v1::find_first_of</td>
<td>Searches for any one of a set of elements.</td>
<td><code>&lt;hpx/include/parallel_find.hpp&gt;</code></td>
<td><code>find_first_of&lt;sup&gt;27&lt;/sup&gt;</code></td>
</tr>
<tr>
<td>hpx::parallel::v1::find_if</td>
<td>Finds the first element satisfying a specific criteria.</td>
<td><code>&lt;hpx/include/parallel_find.hpp&gt;</code></td>
<td><code>find&lt;sup&gt;28&lt;/sup&gt;</code></td>
</tr>
<tr>
<td>hpx::parallel::v1::find_if_not</td>
<td>Finds the first element not satisfying a specific criteria.</td>
<td><code>&lt;hpx/include/parallel_find.hpp&gt;</code></td>
<td><code>find_if_not&lt;sup&gt;29&lt;/sup&gt;</code></td>
</tr>
<tr>
<td>hpx::parallel::v1::for_each</td>
<td>Applies a function to a range of elements.</td>
<td><code>&lt;hpx/include/parallel_for_each.hpp&gt;</code></td>
<td><code>for_each&lt;sup&gt;30&lt;/sup&gt;</code></td>
</tr>
<tr>
<td>hpx::parallel::v1::for_each_n</td>
<td>Applies a function to a number of elements.</td>
<td><code>&lt;hpx/include/parallel_for_each.hpp&gt;</code></td>
<td><code>for_each_n&lt;sup&gt;31&lt;/sup&gt;</code></td>
</tr>
<tr>
<td>hpx::parallel::v1::inclusive_scan</td>
<td>Does an inclusive parallel scan over a range of elements.</td>
<td><code>&lt;hpx/include/parallel_scan.hpp&gt;</code></td>
<td><code>inclusive_scan&lt;sup&gt;32&lt;/sup&gt;</code></td>
</tr>
<tr>
<td>hpx::parallel::v1::lexicographical_compare</td>
<td>Checks if a range of values is lexicographically less than another range of values.</td>
<td><code>&lt;hpx/include/parallel_lexicographical_compare.hpp&gt;</code></td>
<td><code>lexicographical_compare&lt;sup&gt;13&lt;/sup&gt;</code></td>
</tr>
<tr>
<td>hpx::parallel::v1::mismatch</td>
<td>Finds the first position where two ranges differ.</td>
<td><code>&lt;hpx/include/parallel_mismatch.hpp&gt;</code></td>
<td><code>mismatch&lt;sup&gt;34&lt;/sup&gt;</code></td>
</tr>
<tr>
<td>hpx::parallel::v1::none_of</td>
<td>Checks if a predicate is true for none of the elements in a range.</td>
<td><code>&lt;hpx/include/parallel_all_any_none.hpp&gt;</code></td>
<td><code>all_any_none_of</code>&lt;sup&gt;15&lt;/sup&gt;</td>
</tr>
<tr>
<td>hpx::parallel::v1::search</td>
<td>Searches for a range of elements.</td>
<td><code>&lt;hpx/include/parallel_search.hpp&gt;</code></td>
<td><code>search&lt;sup&gt;36&lt;/sup&gt;</code></td>
</tr>
<tr>
<td>hpx::parallel::v1::search_n</td>
<td>Searches for a number consecutive copies of an element in a range.</td>
<td><code>&lt;hpx/include/parallel_search.hpp&gt;</code></td>
<td><code>search_n&lt;sup&gt;14&lt;/sup&gt;</code></td>
</tr>
</tbody>
</table>
Chapter 2. What’s so special about HPX?
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>In header</th>
<th>Algorithm page at cppreference.com</th>
</tr>
</thead>
<tbody>
<tr>
<td>hpx::parallel::copy</td>
<td>Copies a range of elements to a new location.</td>
<td>&lt;hpx/include/parallel_copy.hpp&gt;</td>
<td>exclusive_scan&lt;sup&gt;138&lt;/sup&gt;</td>
</tr>
<tr>
<td>hpx::parallel::copy_n</td>
<td>Copies a number of elements to a new location.</td>
<td>&lt;hpx/include/parallel_copy.hpp&gt;</td>
<td>copy_n&lt;sup&gt;139&lt;/sup&gt;</td>
</tr>
<tr>
<td>hpx::parallel::copy_if</td>
<td>Copies the elements from a range to a new location for which the given predicate is true</td>
<td>&lt;hpx/include/parallel_copy.hpp&gt;</td>
<td>copy&lt;sup&gt;140&lt;/sup&gt;</td>
</tr>
<tr>
<td>hpx::parallel::move</td>
<td>Moves a range of elements to a new location.</td>
<td>&lt;hpx/include/parallel_fill.hpp&gt;</td>
<td>move&lt;sup&gt;141&lt;/sup&gt;</td>
</tr>
<tr>
<td>hpx::parallel::fill</td>
<td>Assigns a range of elements a certain value.</td>
<td>&lt;hpx/include/parallel_fill.hpp&gt;</td>
<td>fill&lt;sup&gt;142&lt;/sup&gt;</td>
</tr>
<tr>
<td>hpx::parallel::fill_n</td>
<td>Assigns a value to a number of elements.</td>
<td>&lt;hpx/include/parallel_fill.hpp&gt;</td>
<td>fill_n&lt;sup&gt;143&lt;/sup&gt;</td>
</tr>
<tr>
<td>hpx::parallel::generate</td>
<td>Saves the result of a function in a range.</td>
<td>&lt;hpx/include/parallel_generate.hpp&gt;</td>
<td>generate&lt;sup&gt;144&lt;/sup&gt;</td>
</tr>
<tr>
<td>hpx::parallel::generate_n</td>
<td>Saves the result of N applications of a function.</td>
<td>&lt;hpx/include/parallel_generate.hpp&gt;</td>
<td>generate_n&lt;sup&gt;145&lt;/sup&gt;</td>
</tr>
<tr>
<td>hpx::parallel::remove</td>
<td>Removes the elements from a range that are equal to the given value.</td>
<td>&lt;hpx/include/parallel_remove.hpp&gt;</td>
<td>remove&lt;sup&gt;146&lt;/sup&gt;</td>
</tr>
<tr>
<td>hpx::parallel::remove</td>
<td>Removes the elements from a range that are equal to the given predicate is false</td>
<td>&lt;hpx/include/parallel_remove.hpp&gt;</td>
<td>remove&lt;sup&gt;147&lt;/sup&gt;</td>
</tr>
<tr>
<td>hpx::parallel::remove_copy</td>
<td>Copies the elements from a range to a new location that are not equal to the given value.</td>
<td>&lt;hpx/include/parallel_remove.hpp&gt;</td>
<td>remove_copy&lt;sup&gt;148&lt;/sup&gt;</td>
</tr>
<tr>
<td>hpx::parallel::remove_copy</td>
<td>Copies the elements from a range to a new location for which the given predicate is false</td>
<td>&lt;hpx/include/parallel_remove.hpp&gt;</td>
<td>remove_copy&lt;sup&gt;149&lt;/sup&gt;</td>
</tr>
<tr>
<td>hpx::parallel::replace</td>
<td>Replaces all values satisfying specific criteria with another value.</td>
<td>&lt;hpx/include/parallel_replace.hpp&gt;</td>
<td>replace&lt;sup&gt;150&lt;/sup&gt;</td>
</tr>
<tr>
<td>hpx::parallel::replace</td>
<td>Replaces all values satisfying specific criteria with another value.</td>
<td>&lt;hpx/include/parallel_replace.hpp&gt;</td>
<td>replace&lt;sup&gt;151&lt;/sup&gt;</td>
</tr>
<tr>
<td>hpx::parallel::replace_copy</td>
<td>Copies a range, replacing elements satisfying specific criteria with another value.</td>
<td>&lt;hpx/include/parallel_replace.hpp&gt;</td>
<td>replace_copy&lt;sup&gt;152&lt;/sup&gt;</td>
</tr>
<tr>
<td>hpx::parallel::replace_copy</td>
<td>Copies a range, replacing elements satisfying specific criteria with another value.</td>
<td>&lt;hpx/include/parallel_replace.hpp&gt;</td>
<td>replace_copy&lt;sup&gt;153&lt;/sup&gt;</td>
</tr>
<tr>
<td>hpx::parallel::reverse</td>
<td>Reverses the order elements in a range.</td>
<td>&lt;hpx/include/parallel_reverse.hpp&gt;</td>
<td>reverse&lt;sup&gt;154&lt;/sup&gt;</td>
</tr>
</tbody>
</table>

2.5. Manual

hpx::parallel::reverse_copy Creates a copy of a range that is reversed.
<hpx/include/parallel_reverse.hpp> reverse_copy<sup>155</sup>
Table 2.17: Set operations on sorted sequences (In Header: `<hpx/include/parallel_algorithm.hpp>`)

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>In header</th>
<th>Algorithm page</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>hpx::parallel::v1::merge</code></td>
<td>Merges two sorted ranges.</td>
<td><code>&lt;hpx/include/parallel_merge.hpp&gt;</code></td>
<td><code>merge</code>&lt;sup&gt;162&lt;/sup&gt;</td>
</tr>
<tr>
<td><code>hpx::parallel::v1::inplace_merge</code></td>
<td>Merges two ordered ranges in-place.</td>
<td><code>&lt;hpx/include/parallel_merge.hpp&gt;</code></td>
<td><code>inplace_merge</code>&lt;sup&gt;163&lt;/sup&gt;</td>
</tr>
<tr>
<td><code>hpx::parallel::v1::includes</code></td>
<td>Returns true if one set is a subset of another.</td>
<td><code>&lt;hpx/include/parallel_set_operations.hpp&gt;</code></td>
<td><code>includes</code>&lt;sup&gt;164&lt;/sup&gt;</td>
</tr>
<tr>
<td><code>hpx::parallel::v1::set_difference</code></td>
<td>Computes the difference between two sets.</td>
<td><code>&lt;hpx/include/parallel_set_operations.hpp&gt;</code></td>
<td><code>set_difference</code>&lt;sup&gt;165&lt;/sup&gt;</td>
</tr>
<tr>
<td><code>hpx::parallel::v1::set_intersection</code></td>
<td>Computes the intersection of two sets.</td>
<td><code>&lt;hpx/include/parallel_set_operations.hpp&gt;</code></td>
<td><code>set_intersection</code>&lt;sup&gt;166&lt;/sup&gt;</td>
</tr>
<tr>
<td><code>hpx::parallel::v1::set_symmetric_difference</code></td>
<td>Computes the symmetric difference between two sets.</td>
<td><code>&lt;hpx/include/parallel_set_operations.hpp&gt;</code></td>
<td><code>set_symmetric_difference</code>&lt;sup&gt;167&lt;/sup&gt;</td>
</tr>
<tr>
<td><code>hpx::parallel::v1::set_union</code></td>
<td>Computes the union of two sets.</td>
<td><code>&lt;hpx/include/parallel_set_operations.hpp&gt;</code></td>
<td><code>set_union</code>&lt;sup&gt;168&lt;/sup&gt;</td>
</tr>
</tbody>
</table>

139 http://en.cppreference.com/w/cpp/algorithm/copy_n  
143 http://en.cppreference.com/w/cpp/algorithm/fill_n  
144 http://en.cppreference.com/w/cpp/algorithm/generate  
146 http://en.cppreference.com/w/cpp/algorithm/remove  
147 http://en.cppreference.com/w/cpp/algorithm/remove_copy  
149 http://en.cppreference.com/w/cpp/algorithm/remove_copy  
159 http://en.cppreference.com/w/cpp/algorithm/rotate_copy  
165 http://en.cppreference.com/w/cpp/algorithm/includes  
### Table 2.18: Heap operations (In Header: `<hpx/include/parallel_algorithm.hpp>`)

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>In header</th>
<th>Algorithm page at cppreference.com</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>hpx::parallel::v1::is_heap</code></td>
<td>Returns true if the range is max heap.</td>
<td><code>&lt;hpx/include/is_heap.hpp&gt;</code></td>
<td>is_heap[^169]</td>
</tr>
<tr>
<td><code>hpx::parallel::v1::is_heap_until</code></td>
<td>Returns the first element that breaks a max heap.</td>
<td><code>&lt;hpx/include/is_heap.hpp&gt;</code></td>
<td>is_heap_until[^170]</td>
</tr>
</tbody>
</table>


### Table 2.19: Minimum/maximum operations (In Header: `<hpx/include/parallel_algorithm.hpp>`)

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>In header</th>
<th>Algorithm page at cppreference.com</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>hpx::parallel::v1::max_element</code></td>
<td>Returns the largest element in a range.</td>
<td><code>&lt;hpx/include/parallel_minmax.hpp&gt;</code></td>
<td>max_element[^171]</td>
</tr>
<tr>
<td><code>hpx::parallel::v1::min_element</code></td>
<td>Returns the smallest element in a range.</td>
<td><code>&lt;hpx/include/parallel_minmax.hpp&gt;</code></td>
<td>min_element[^172]</td>
</tr>
<tr>
<td><code>hpx::parallel::v1::minmax_element</code></td>
<td>Returns the smallest and the largest element in a range.</td>
<td><code>&lt;hpx/include/parallel_minmax.hpp&gt;</code></td>
<td>minmax_element[^173]</td>
</tr>
</tbody>
</table>


### Table 2.20: Partitioning Operations (In Header: `<hpx/include/parallel_algorithm.hpp>`)

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>In header</th>
<th>Algorithm page at cppreference.com</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>hpx::parallel::v1::is_partitioned</code></td>
<td>Returns true if each true element for a predicate precedes the false elements in a range.</td>
<td><code>&lt;hpx/include/parallel_is_partitioned.hpp&gt;</code></td>
<td>is_partitioned[^174]</td>
</tr>
<tr>
<td><code>hpx::parallel::v1::partition</code></td>
<td>Divides elements into two groups without preserving their relative order.</td>
<td><code>&lt;hpx/include/parallel_partition.hpp&gt;</code></td>
<td>partition[^175]</td>
</tr>
<tr>
<td><code>hpx::parallel::v1::partition_copy</code></td>
<td>Copies a range dividing the elements into two groups.</td>
<td><code>&lt;hpx/include/parallel_partition.hpp&gt;</code></td>
<td>partition_copy[^176]</td>
</tr>
<tr>
<td><code>hpx::parallel::v1::stable_partition</code></td>
<td>Divides elements into two groups while preserving their relative order.</td>
<td><code>&lt;hpx/include/parallel_partition.hpp&gt;</code></td>
<td>stable_partition[^177]</td>
</tr>
</tbody>
</table>

### Table 2.21: Sorting Operations (In Header: `<hpx/include/parallel_algorithm.hpp>`)  

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>In header</th>
<th>Algorithm page at cppreference.com</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>hpx::parallel::v1::is_sorted</code></td>
<td>Returns true if each element in a range is sorted.</td>
<td><code>&lt;hpx/include/parallel_is_sorted.hpp&gt;</code></td>
<td><a href="http://en.cppreference.com/w/cpp/algorithm/is_sorted">is_sorted</a></td>
</tr>
<tr>
<td><code>hpx::parallel::v1::is_sorted_until</code></td>
<td>Returns the first unsorted element.</td>
<td><code>&lt;hpx/include/parallel_is_sorted.hpp&gt;</code></td>
<td><a href="http://en.cppreference.com/w/cpp/algorithm/is_sorted_until">is_sorted_until</a></td>
</tr>
<tr>
<td><code>hpx::parallel::v1::sort</code></td>
<td>Sorts the elements in a range.</td>
<td><code>&lt;hpx/include/parallel_sort.hpp&gt;</code></td>
<td><a href="http://en.cppreference.com/w/cpp/algorithm/sort">sort</a></td>
</tr>
<tr>
<td><code>hpx::parallel::v1::sort_by_key</code></td>
<td>Sorts one range of data using keys supplied in another range.</td>
<td><code>&lt;hpx/include/parallel_sort.hpp&gt;</code></td>
<td></td>
</tr>
</tbody>
</table>

### Table 2.22: Numeric Parallel Algorithms (In Header: `<hpx/include/parallel_numeric.hpp>`)  

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>In header</th>
<th>Algorithm page at cppreference.com</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>hpx::parallel::v1::adjacent_difference</code></td>
<td>Calculates the difference between each element in an input range and the preceding element.</td>
<td><code>&lt;hpx/include/parallel_adjacent_difference.hpp&gt;</code></td>
<td><a href="http://en.cppreference.com/w/cpp/algorithm/adjacent_difference">adjacent_difference</a></td>
</tr>
<tr>
<td><code>hpx::parallel::v1::reduce</code></td>
<td>Sums up a range of elements.</td>
<td><code>&lt;hpx/include/parallel_reduce.hpp&gt;</code></td>
<td><a href="http://en.cppreference.com/w/cpp/algorithm/reduce">reduce</a></td>
</tr>
<tr>
<td><code>hpx::parallel::v1::reduce_by_key</code></td>
<td>Performs an inclusive scan on consecutive elements with matching keys, with a reduction to output only the final sum for each key. The key sequence <code>{1,1,1,2,3,3,3,3,1}</code> and value sequence <code>{2,3,4,5,6,7,8,9,10}</code> would be reduced to keys=<code>{1,2,3,1}</code>, values=<code>{9,5,30,10}</code>.</td>
<td><code>&lt;hpx/include/parallel_reduce.hpp&gt;</code></td>
<td></td>
</tr>
<tr>
<td><code>hpx::parallel::v1::transform_reduce</code></td>
<td>Sums up a range of elements after applying a function. Also, accumulates the inner products of two input ranges.</td>
<td><code>&lt;hpx/include/parallel_transform_reduce.hpp&gt;</code></td>
<td><a href="http://en.cppreference.com/w/cpp/algorithm/transform_reduce">transform_reduce</a></td>
</tr>
<tr>
<td><code>hpx::parallel::v1::transform_inclusive_scan</code></td>
<td>Does an inclusive parallel scan over a range of elements after applying a function.</td>
<td><code>&lt;hpx/include/parallel_scan.hpp&gt;</code></td>
<td><a href="http://en.cppreference.com/w/cpp/algorithm/transform_inclusive_scan">transform_inclusive_scan</a></td>
</tr>
<tr>
<td><code>hpx::parallel::v1::transform_exclusive_scan</code></td>
<td>Does an exclusive parallel scan over a range of elements after applying a function.</td>
<td><code>&lt;hpx/include/parallel_scan.hpp&gt;</code></td>
<td><a href="http://en.cppreference.com/w/cpp/algorithm/transform_exclusive_scan">transform_exclusive_scan</a></td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
<td>In header</td>
<td>Algorithm page at cppreference.com</td>
</tr>
<tr>
<td>------</td>
<td>-------------</td>
<td>-----------</td>
<td>------------------------------------</td>
</tr>
<tr>
<td>hpx::parallel::v1::destroy</td>
<td>Destroys a range of objects.</td>
<td>&lt;hpx/include/parallel_destroy.hpp&gt;</td>
<td>destroy&lt;sup&gt;180&lt;/sup&gt;</td>
</tr>
<tr>
<td>hpx::parallel::v1::destroy</td>
<td>Destroys a range of objects.</td>
<td>&lt;hpx/include/parallel_destroy.hpp&gt;</td>
<td>destroy&lt;sup&gt;180&lt;/sup&gt;</td>
</tr>
<tr>
<td>hpx::parallel::v1::uninitialized_copy</td>
<td>Copies a range of objects to an uninitialized area of memory.</td>
<td>&lt;hpx/include/parallel_uninitialized_copy.hpp&gt;</td>
<td>uninitialized_copy&lt;sup&gt;188&lt;/sup&gt;</td>
</tr>
<tr>
<td>hpx::parallel::v1::uninitialized_copy</td>
<td>Copies a range of objects to an uninitialized area of memory.</td>
<td>&lt;hpx/include/parallel_uninitialized_copy.hpp&gt;</td>
<td>uninitialized_copy&lt;sup&gt;189&lt;/sup&gt;</td>
</tr>
<tr>
<td>hpx::parallel::v1::uninitialized_move</td>
<td>Moves a range of objects to an uninitialized area of memory.</td>
<td>&lt;hpx/include/parallel_uninitialized_move.hpp&gt;</td>
<td>uninitialized_move&lt;sup&gt;194&lt;/sup&gt;</td>
</tr>
<tr>
<td>hpx::parallel::v1::uninitialized_move</td>
<td>Moves a range of objects to an uninitialized area of memory.</td>
<td>&lt;hpx/include/parallel_uninitialized_move.hpp&gt;</td>
<td>uninitialized_move&lt;sup&gt;195&lt;/sup&gt;</td>
</tr>
<tr>
<td>hpx::parallel::v1::uninitialized_value_construct</td>
<td>Creates objects in an uninitialized area of memory.</td>
<td>&lt;hpx/include/parallel_uninitialized_value_construct.hpp&gt;</td>
<td>uninitialized_value_construct&lt;sup&gt;196&lt;/sup&gt;</td>
</tr>
<tr>
<td>hpx::parallel::v1::uninitialized_value_construct</td>
<td>Creates objects in an uninitialized area of memory.</td>
<td>&lt;hpx/include/parallel_uninitialized_value_construct.hpp&gt;</td>
<td>uninitialized_value_construct&lt;sup&gt;197&lt;/sup&gt;</td>
</tr>
</tbody>
</table>


2.5. Manual 145
Table 2.24: Index-based for-loops (In Header: `<hpx/include/parallel_algorithm.hpp>`)

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>In header</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>hpx::parallel::v2::for_loop</code></td>
<td>Implements loop functionality over a range specified by integral or iterator bounds.</td>
<td><code>&lt;hpx/include/parallel_for_loop.hpp&gt;</code></td>
</tr>
<tr>
<td><code>hpx::parallel::v2::for_loop_strided</code></td>
<td>Implements loop functionality over a range specified by integral or iterator bounds.</td>
<td><code>&lt;hpx/include/parallel_for_loop.hpp&gt;</code></td>
</tr>
<tr>
<td><code>hpx::parallel::v2::for_loop_n</code></td>
<td>Implements loop functionality over a range specified by integral or iterator bounds.</td>
<td><code>&lt;hpx/include/parallel_for_loop.hpp&gt;</code></td>
</tr>
<tr>
<td><code>hpx::parallel::v2::for_loop_n_strided</code></td>
<td>Implements loop functionality over a range specified by integral or iterator bounds.</td>
<td><code>&lt;hpx/include/parallel_for_loop.hpp&gt;</code></td>
</tr>
</tbody>
</table>

Executor parameters and executor parameter traits

`HPX` introduces the notion of execution parameters and execution parameter traits. At this point, the only parameter that can be customized is the size of the chunks of work executed on a single `HPX` thread (such as the number of loop iterations combined to run as a single task).

An executor parameter object is responsible for exposing the calculation of the size of the chunks scheduled. It abstracts the (potentially platform-specific) algorithms of determining those chunk sizes.

The way executor parameters are implemented is aligned with the way executors are implemented. All functionalities of concrete executor parameter types are exposed and accessible through a corresponding `hpx::parallel::executor_parameter_traits` type.

With `executor_parameter_traits`, clients access all types of executor parameters uniformly:

```cpp
std::size_t chunk_size =
    executor_parameter_traits<my_parameter_t>::get_chunk_size(my_parameter,
        my_executor, [](){ return 0; }, num_tasks);
```

This call synchronously retrieves the size of a single chunk of loop iterations (or similar) to combine for execution on a single `HPX` thread if the overall number of tasks to schedule is given by `num_tasks`. The lambda function exposes a means of test-probing the execution of a single iteration for performance measurement purposes. The execution parameter type might dynamically determine the execution time of one or more tasks in order to calculate the chunk size; see `hpx::parallel::execution::auto_chunk_size` for an example of this executor parameter type.

Other functions in the interface exist to discover whether an executor parameter type should be invoked once (i.e., it returns a static chunk size; see `hpx::parallel::execution::static_chunk_size`) or whether it should be invoked for each scheduled chunk of work (i.e., it returns a variable chunk size; for an example, see `hpx::parallel::execution::guided_chunk_size`).

188 http://en.cppreference.com/w/cpp/memory/ uninitialized_copy
189 http://en.cppreference.com/w/cpp/memory/ uninitialized_copy_n
190 http://en.cppreference.com/w/cpp/memory/ uninitialized_default_construct
191 http://en.cppreference.com/w/cpp/memory/ uninitialized_default_construct_n
193 http://en.cppreference.com/w/cpp/memory/ uninitialized_fill_n
197 http://en.cppreference.com/w/cpp/memory/ uninitialized_value_construct_n
Although this interface appears to require executor parameter type authors to implement all different basic operations, none are required. In practice, all operations have sensible defaults. However, some executor parameter types will naturally specialize all operations for maximum efficiency.

**HPX** implements the following executor parameter types:

- **hpx::parallel::execution::auto_chunk_size**: Loop iterations are divided into pieces and then assigned to threads. The number of loop iterations combined is determined based on measurements of how long the execution of 1% of the overall number of iterations takes. This executor parameter type makes sure that as many loop iterations are combined as necessary to run for the amount of time specified.

- **hpx::parallel::execution::static_chunk_size**: Loop iterations are divided into pieces of a given size and then assigned to threads. If the size is not specified, the iterations are, if possible, evenly divided contiguously among the threads. This executor parameters type is equivalent to OpenMP’s STATIC scheduling directive.

- **hpx::parallel::execution::dynamic_chunk_size**: Loop iterations are divided into pieces of a given size and then dynamically scheduled among the cores; when a core finishes one chunk, it is dynamically assigned another. If the size is not specified, the default chunk size is 1. This executor parameter type is equivalent to OpenMP’s DYNAMIC scheduling directive.

- **hpx::parallel::execution::guided_chunk_size**: Iterations are dynamically assigned to cores in blocks as cores request them until no blocks remain to be assigned. This is similar to dynamic_chunk_size except that the block size decreases each time a number of loop iterations is given to a thread. The size of the initial block is proportional to `number_of_iterations / number_of_cores`. Subsequent blocks are proportional to `number_of_iterations_remaining / number_of_cores`. The optional chunk size parameter defines the minimum block size. The default minimal chunk size is 1. This executor parameter type is equivalent to OpenMP’s GUIDED scheduling directive.

### Using task blocks

The `define_task_block`, `run` and the `wait` functions implemented based on N4411¹⁹⁸ are based on the `task_block` concept that is a part of the common subset of the Microsoft Parallel Patterns Library (PPL)¹⁹⁹ and the Intel Threading Building Blocks (TBB)²⁰⁰ libraries.

These implementations adopt a simpler syntax than exposed by those libraries— one that is influenced by language-based concepts, such as spawn and sync from Cilk++²⁰¹ and async and finish from X10²⁰². They improve on existing practice in the following ways:

- The exception handling model is simplified and more consistent with normal C++ exceptions.
- Most violations of strict fork-join parallelism can be enforced at compile time (with compiler assistance, in some cases).
- The syntax allows scheduling approaches other than child stealing.

Consider an example of a parallel traversal of a tree, where a user-provided function `compute` is applied to each node of the tree, returning the sum of the results:

```cpp
template <typename Func>
int traverse(node& n, Func && compute)
{
    int left = 0, right = 0;
    define_task_block(
```

²⁰⁰ https://www.threadingbuildingblocks.org/
²⁰² https://x10-lang.org/
The example above demonstrates the use of two of the functions, `hpx::parallel::define_task_block` and the `hpx::parallel::task_block::run` member function of a `hpx::parallel::task_block`.

The `task_block` function delineates a region in a program code potentially containing invocations of threads spawned by the `run` member function of the `task_block` class. The `run` function spawns an HPX thread, a unit of work that is allowed to execute in parallel with respect to the caller. Any parallel tasks spawned by `run` within the task block are joined back to a single thread of execution at the end of the `define_task_block`. `run` takes a user-provided function object `f` and starts it asynchronously—i.e., it may return before the execution of `f` completes. The HPX scheduler may choose to run `f` immediately or delay running `f` until compute resources become available.

A `task_block` can be constructed only by `define_task_block` because it has no public constructors. Thus, `run` can be invoked directly or indirectly only from a user-provided function passed to `define_task_block`:

```cpp
void g();

void f(task_block<> & tr)
{
    tr.run(g); // OK, invoked from within task_block in h
}

void h()
{
    define_task_block(f);
}

int main()
{
    task_block<> tr; // Error: no public constructor
    tr.run(g); // No way to call run outside of a define_task_block
    return 0;
}
```

### Extensions for task blocks

#### Using execution policies with task blocks

HPX implements some extensions for `task_block` beyond the actual standards proposal N4411[^203]. The main addition is that a `task_block` can be invoked with an execution policy as its first argument, very similar to the parallel algorithms.

An execution policy is an object that expresses the requirements on the ordering of functions invoked as a consequence of the invocation of a task block. Enabling passing an execution policy to `define_task_block` gives the user control over the amount of parallelism employed by the created `task_block`. In the following example the use of an explicit `par` execution policy makes the user’s intent explicit:

template <typename Func>
int traverse(node *n, Func&& compute)
{
    int left = 0, right = 0;

    define_task_block(
        execution::par,                       // execution::parallel_policy
        [n](task_block<> & tb) {
            if (n->left)
                tb.run([=] { left = traverse(n->left, compute); });
            if (n->right)
                tb.run([=] { right = traverse(n->right, compute); });
        });
    return compute(n) + left + right;
}

This also causes the `hpx::parallel::v2::task_block` object to be a template in our implementation. The template argument is the type of the execution policy used to create the task block. The template argument defaults to `hpx::parallel::execution::parallel_policy`.

HPX still supports calling `hpx::parallel::v2::define_task_block` without an explicit execution policy. In this case the task block will run using the `hpx::parallel::execution::parallel_policy`.

HPX also adds the ability to access the execution policy that was used to create a given `task_block`.

### Using executors to run tasks

Often, users want to be able to not only define an execution policy to use by default for all spawned tasks inside the task block, but also to customize the execution context for one of the tasks executed by `task_block::run`. Adding an optionally passed executor instance to that function enables this use case:

```cpp
template <typename Func>
int traverse(node *n, Func&& compute)
{
    int left = 0, right = 0;

    define_task_block(
        execution::par,                       // execution::parallel_policy
        [n](auto & tb) {
            if (n->left)
            {
                // use explicitly specified executor to run this task
                tb.run(my_executor(), [=] { left = traverse(n->left, compute); });
            }
            if (n->right)
            {
                // use the executor associated with the par execution policy
                tb.run([n] { right = traverse(n->right, compute); });
            }
        });
    return compute(n) + left + right;
}
```

HPX still supports calling `hpx::parallel::v2::task_block::run` without an explicit executor object. In this case the task will be run using the executor associated with the execution policy that was used to call `hpx::parallel::v2::define_task_block`.

2.5. Manual 149
2.5.7 Writing distributed HPX applications

This section focuses on the features of HPX needed to write distributed applications, namely the Active Global Address Space (AGAS), remotely executable functions (i.e. actions), and distributed objects (i.e. components).

Global names

HPX implements an Active Global Address Space (AGAS) which is exposing a single uniform address space spanning all localities an application runs on. AGAS is a fundamental component of the ParalleX execution model. Conceptually, there is no rigid demarcation of local or global memory in AGAS; all available memory is a part of the same address space. AGAS enables named objects to be moved (migrated) across localities without having to change the object’s name, i.e., no references to migrated objects have to be ever updated. This feature has significance for dynamic load balancing and in applications where the workflow is highly dynamic, allowing work to be migrated from heavily loaded nodes to less loaded nodes. In addition, immutability of names ensures that AGAS does not have to keep extra indirections (“bread crumbs”) when objects move, hence minimizing complexity of code management for system developers as well as minimizing overheads in maintaining and managing aliases.

The AGAS implementation in HPX does not automatically expose every local address to the global address space. It is the responsibility of the programmer to explicitly define which of the objects have to be globally visible and which of the objects are purely local.

In HPX global addresses (global names) are represented using the hpx::id_type data type. This data type is conceptually very similar to void* pointers as it does not expose any type information of the object it is referring to.

The only predefined global addresses are assigned to all localities. The following HPX API functions allow one to retrieve the global addresses of localities:

- \texttt{hpx::find\_here}: retrieve the global address of the locality this function is called on.
- \texttt{hpx::find\_all\_localities}: retrieve the global addresses of all localities available to this application (including the locality the function is being called on).
- \texttt{hpx::find\_remote\_localities}: retrieve the global addresses of all remote localities available to this application (not including the locality the function is being called on)
- \texttt{hpx::get\_num\_localities}: retrieve the number of localities available to this application.
- \texttt{hpx::find\_locality}: retrieve the global address of any locality supporting the given component type.
- \texttt{hpx::get\_colocation\_id}: retrieve the global address of the locality currently hosting the object with the given global address.

Additionally, the global addresses of localities can be used to create new instances of components using the following HPX API function:

- \texttt{hpx::components::new\_:} Create a new instance of the given Component type on the specified locality.

\textbf{Note}: HPX does not expose any functionality to delete component instances. All global addresses (as represented using hpx::id_type) are automatically garbage collected. When the last (global) reference to a particular component instance goes out of scope the corresponding component instance is automatically deleted.

Applying actions

Action type definition

Actions are special types we use to describe possibly remote operations. For every global function and every member function which has to be invoked distantly, a special type must be defined. For any global function the special macro HPX\_PLAIN\_ACTION can be used to define the action type. Here is an example demonstrating this:
namespace app
{
    void some_global_function(double d)
    {
        cout << d;
    }
}

// This will define the action type 'some_global_action' which represents
// the function 'app::some_global_function'.
HPX_PLAIN_ACTION(app::some_global_function, some_global_action);

Important: The macro HPX_PLAIN_ACTION has to be placed in global namespace, even if the wrapped function
is located in some other namespace. The newly defined action type is placed in the global namespace as well.

If the action type should be defined somewhere not in global namespace, the action type definition has to be split into
two macro invocations (HPX_DEFINE_PLAIN_ACTION and HPX_REGISTER_ACTION) as shown in the next example:

namespace app
{
    void some_global_function(double d)
    {
        cout << d;
    }

    // On conforming compilers the following macro expands to:
    // typedef hpx::actions::make_action<
    //    decltype(&some_global_function), &some_global_function
    // >::type some_global_action;

    HPX_DEFINE_PLAIN_ACTION(some_global_function, some_global_action);
}

// The following macro expands to a series of definitions of global objects
// which are needed for proper serialization and initialization support
// enabling the remote invocation of the function 'some_global_function'.
HPX_REGISTER_ACTION(app::some_global_action, app_some_global_action);

The shown code defines an action type some_global_action inside the namespace app.

Important: If the action type definition is split between two macros as shown above, the name of the action type to
create has to be the same for both macro invocations (here some_global_action).

Important: The second argument passed to HPX_REGISTER_ACTION (app_some_global_action) has to
comprise a globally unique C++ identifier representing the action. This is used for serialization purposes.

For member functions of objects which have been registered with AGAS (e.g. ‘components’) a different registration
macro HPX_DEFINE_COMPONENT_ACTION has to be utilized. Any component needs to be declared in a header

2.5. Manual
file and have some special support macros defined in a source file. Here is an example demonstrating this. The first snippet has to go into the header file:

```cpp
namespace app
{
  struct some_component
  :
      hpx::components::component_base<some_component>
  {
    int some_member_function(std::string s)
    {
      return boost::lexical_cast<int>(s);
    }

    // This will define the action type 'some_member_action' which
    // represents the member function 'some_member_function' of the
    // object type 'some_component'.
    HPX_DEFINE_COMPONENT_ACTION(some_component, some_member_function, some_member_action);
  }
}
```

// Note: The second argument to the macro below has to be systemwide-unique
// C++ identifiers
HPX_REGISTER_ACTION_DECLARATION(app::some_component::some_member_action, some_component_some_action);

The next snippet belongs into a source file (e.g. the main application source file) in the simplest case:

```cpp
typedef hpx::components::component<app::some_component> component_type;
typedef app::some_component some_component;

HPX_REGISTER_COMPONENT(component_type, some_component);

// The parameters for this macro have to be the same as used in the corresponding
// HPX_REGISTER_ACTION_DECLARATION() macro invocation above
typedef some_component::some_member_action some_component_some_action;
HPX_REGISTER_ACTION(some_component_some_action);
```

Granted, these macro invocations are a bit more complex than for simple global functions, however we believe they are still manageable.

The most important macro invocation is the `HPX_DEFINE_COMPONENT_ACTION` in the header file as this defines the action type we need to invoke the member function. For a complete example of a simple component action see [hpx_link examples/quickstart/component_in_executable.cpp..component_in_executable.cpp](#)

## Action invocation

The process of invoking a global function (or a member function of an object) with the help of the associated action is called ‘applying the action’. Actions can have arguments, which will be supplied while the action is applied. At the minimum, one parameter is required to apply any action - the id of the locality the associated function should be invoked on (for global functions), or the id of the component instance (for member functions). Generally, HPX provides several ways to apply an action, all of which are described in the following sections.

Generally, HPX actions are very similar to ‘normal’ C++ functions except that actions can be invoked remotely. Fig. ?? below shows an overview of the main API exposed by HPX. This shows the function invocation syntax as defined by the C++ language (dark gray), the additional invocation syntax as provided through C++ Standard Library features (medium gray), and the extensions added by HPX (light gray) where:

- f function to invoke,
• p... (optional) arguments,
• R: return type of f,
• action: action type defined by, \texttt{HPX\_DEFINE\_PLAIN\_ACTION} or \texttt{HPX\_DEFINE\_COMPONENT\_ACTION} encapsulating f,
• a: an instance of the type `action,
• id: the global address the action is applied to.

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|}
\hline
R f(p...) & Synchronous Execution & Asynchronous Execution & Fire & Forget Execution \\
& (returns \(R\)) & (returns future<\(R\)>) & (returns void) \\
\hline
Functions & \(f(p...)\) & \texttt{async}(f, p...) & \texttt{apply}(f, \ldots) \\
(direct invocation) & & \texttt{C++} & \\
\hline
Functions & \texttt{bind}(f, p...)(\ldots) & \texttt{async}(\texttt{bind}(f, p...), \ldots) & \texttt{apply}(\texttt{bind}(f, p...), \ldots) \\
(lazy invocation) & & \texttt{C++ Standard Library} & \\
\hline
Actions & \texttt{HPX\_ACTION}(f, action) & \texttt{HPX\_ACTION}(f, action) & \texttt{HPX\_ACTION}(f, action) \\
(direct invocation) & a(id, \ldots) & \texttt{async}(a, id, p...) & \texttt{apply}(a, id, p...) \\
\hline
Actions & \texttt{HPX\_ACTION}(f, action) & \texttt{HPX\_ACTION}(f, action) & \texttt{HPX\_ACTION}(f, action) \\
(lazy invocation) & \texttt{bind}(a, id, p...)(\ldots) & \texttt{async}(\texttt{bind}(a, id, p...), \ldots) & \texttt{apply}(\texttt{bind}(a, id, p...), \ldots) \\
\hline
\end{tabular}
\end{table}

Fig. 2.8: Overview of the main API exposed by \texttt{HPX}.

This figure shows that \texttt{HPX} allows the user to apply actions with a syntax similar to the C++ standard. In fact, all action types have an overloaded function operator allowing to synchronously apply the action. Further, \texttt{HPX} implements \texttt{hpx::async} which semantically works similar to the way \texttt{std::async} works for plain C++ function.

\textbf{Note:} The similarity of applying an action to conventional function invocations extends even further. \texttt{HPX} implements \texttt{hpx::bind} and \texttt{hpx::function} two facilities which are semantically equivalent to the \texttt{std::bind} and \texttt{std::function} types as defined by the C++11 Standard. While \texttt{hpx::async} extends beyond the conventional semantics by supporting actions and conventional C++ functions, the \texttt{HPX} facilities \texttt{hpx::bind} and \texttt{hpx::function} extend beyond the conventional standard facilities too. The \texttt{HPX} facilities not only support conventional functions, but can be used for actions as well.

Additionally, \texttt{HPX} exposes \texttt{hpx::apply} and \texttt{hpx::async\_continue} both of which refine and extend the standard C++ facilities.

The different ways to invoke a function in \texttt{HPX} will be explained in more detail in the following sections.

**Applying an action asynchronously without any synchronization**

This method (‘fire and forget’) will make sure the function associated with the action is scheduled to run on the target \texttt{locality}. Applying the action does not wait for the function to start running, instead it is a fully asynchronous operation. The following example shows how to apply the action as defined in the previous section on the local \texttt{locality} (the \texttt{locality} this code runs on):

```cpp
some_global_action act; // define an instance of some_global_action
hpx::apply(act, hpx::find_here(), 2.0);
```

\section*{2.5. Manual}
(the function `hpx::find_here()` returns the id of the local *locality*, i.e. the *locality* this code executes on).

Any component member function can be invoked using the same syntactic construct. Given that `id` is the global address for a component instance created earlier, this invocation looks like:

```cpp
some_component_action act; // define an instance of some_component_action
hpx::apply(act, id, "42");
```

In this case any value returned from this action (e.g. in this case the integer 42 is ignored. Please look at *Action type definition* for the code defining the component action `some_component_action` used.

### Applying an action asynchronously with synchronization

This method will make sure the action is scheduled to run on the target *locality*. Applying the action itself does not wait for the function to start running or to complete, instead this is a fully asynchronous operation similar to using `hpx::apply` as described above. The difference is that this method will return an instance of a `hpx::future<>` encapsulating the result of the (possibly remote) execution. The future can be used to synchronize with the asynchronous operation. The following example shows how to apply the action from above on the local *locality*:

```cpp
some_global_action act; // define an instance of some_global_action
hpx::future<void> f = hpx::async(act, hpx::find_here(), 2.0); // ... other code can be executed here
f.get(); // this will possibly wait for the asynchronous operation to 'return'
```

(as before, the function `hpx::find_here()` returns the id of the local *locality* (the *locality* this code is executed on).

**Note:** The use of a `hpx::future<void>` allows the current thread to synchronize with any remote operation not returning any value.

**Note:** Any `std::future<>` returned from `std::async()` is required to block in its destructor if the value has not been set for this future yet. This is not true for `hpx::future<>` which will never block in its destructor, even if the value has not been returned to the future yet. We believe that consistency in the behavior of futures is more important than standards conformance in this case.

Any component member function can be invoked using the same syntactic construct. Given that `id` is the global address for a component instance created earlier, this invocation looks like:

```cpp
some_component_action act; // define an instance of some_component_action
hpx::future<int> f = hpx::async(act, id, "42"); // ... other code can be executed here
cout << f.get(); // this will possibly wait for the asynchronous operation to 'return' 42
```

**Note:** The invocation of `f.get()` will return the result immediately (without suspending the calling thread) if the result from the asynchronous operation has already been returned. Otherwise, the invocation of `f.get()` will suspend the execution of the calling thread until the asynchronous operation returns its result.
Applying an action synchronously

This method will schedule the function wrapped in the specified action on the target locality. While the invocation appears to be synchronous (as we will see), the calling thread will be suspended while waiting for the function to return. Invoking a plain action (e.g. a global function) synchronously is straightforward:

```cpp
some_global_action act; // define an instance of some_global_action
act(hpx::find_here(), 2.0);
```

While this call looks just like a normal synchronous function invocation, the function wrapped by the action will be scheduled to run on a new thread and the calling thread will be suspended. After the new thread has executed the wrapped global function, the waiting thread will resume and return from the synchronous call.

Equivalently, any action wrapping a component member function can be invoked synchronously as follows:

```cpp
some_component_action act; // define an instance of some_component_action
int result = act(id, "42");
```

The action invocation will either schedule a new thread locally to execute the wrapped member function (as before, id is the global address of the component instance the member function should be invoked on), or it will send a parcel to the remote locality of the component causing a new thread to be scheduled there. The calling thread will be suspended until the function returns its result. This result will be returned from the synchronous action invocation.

It is very important to understand that this ‘synchronous’ invocation syntax in fact conceals an asynchronous function call. This is beneficial as the calling thread is suspended while waiting for the outcome of a potentially remote operation. The HPX thread scheduler will schedule other work in the meantime, allowing the application to make further progress while the remote result is computed. This helps overlapping computation with communication and hiding communication latencies.

Note: The syntax of applying an action is always the same, regardless whether the target locality is remote to the invocation locality or not. This is a very important feature of HPX as it frees the user from the task of keeping track what actions have to be applied locally and which actions are remote. If the target for applying an action is local, a new thread is automatically created and scheduled. Once this thread is scheduled and run, it will execute the function encapsulated by that action. If the target is remote, HPX will send a parcel to the remote locality which encapsulates the action and its parameters. Once the parcel is received on the remote locality HPX will create and schedule a new thread there. Once this thread runs on the remote locality, it will execute the function encapsulated by the action.

Applying an action with a continuation but without any synchronization

This method is very similar to the method described in section Applying an action asynchronously without any synchronization. The difference is that it allows the user to chain a sequence of asynchronous operations, while handing the (intermediate) results from one step to the next step in the chain. Where hpx::apply invokes a single function using ‘fire and forget’ semantics, hpx::apply_continue asynchronously triggers a chain of functions without the need for the execution flow ‘to come back’ to the invocation site. Each of the asynchronous functions can be executed on a different locality.

Applying an action with a continuation and with synchronization

This method is very similar to the method described in section Applying an action asynchronously with synchronization. In addition to what hpx::async can do, the functions hpx::async_continue takes an additional function argument. This function will be called as the continuation of the executed action. It is expected to perform additional operations and to make sure that a result is returned to the original invocation site. This method chains operations asynchronously by providing a continuation operation which is automatically executed once the first action has finished executing.
As an example we chain two actions, where the result of the first action is forwarded to the second action and the result of the second action is sent back to the original invocation site:

```cpp
// first action
std::int32_t action1(std::int32_t i)
{
    return i+1;
}
HPX_PLAIN_ACTION(action1); // defines action1_type

// second action
std::int32_t action2(std::int32_t i)
{
    return i*2;
}
HPX_PLAIN_ACTION(action2); // defines action2_type

// this code invokes 'action1' above and passes along a continuation
// function which will forward the result returned from 'action1' to
// 'action2'.
action1_type act1; // define an instance of 'action1_type'
action2_type act2; // define an instance of 'action2_type'
hpx::future<int> f =
    hpx::async_continue(act1, hpx::make_continuation(act2),
        hpx::find_here(), 42);
hpx::cout << f.get() << "\n"; // will print: 86 ((42 + 1) * 2)
```

By default, the continuation is executed on the same `locality` as `hpx::async_continue` is invoked from. If you want to specify the `locality` where the continuation should be executed, the code above has to be written as:

```cpp
// this code invokes 'action1' above and passes along a continuation
// function which will forward the result returned from 'action1' to
// 'action2'.
action1_type act1; // define an instance of 'action1_type'
action2_type act2; // define an instance of 'action2_type'
hpx::future<int> f =
    hpx::async_continue(act1, hpx::make_continuation(act2, hpx::find_here()),
        hpx::find_here(), 42);
hpx::cout << f.get() << "\n"; // will print: 86 ((42 + 1) * 2)
```

Similarly, it is possible to chain more than 2 operations:

```cpp
action1_type act1; // define an instance of 'action1_type'
action2_type act2; // define an instance of 'action2_type'
hpx::future<int> f =
    hpx::async_continue(act1,
        hpx::make_continuation(act2, hpx::make_continuation(act1)),
        hpx::find_here(), 42);
hpx::cout << f.get() << "\n"; // will print: 87 ((42 + 1) * 2 + 1)
```

The function `hpx::make_continuation` creates a special function object which exposes the following prototype:

```cpp
struct continuation
{
    template <typename Result>
    void operator()(hpx::id_type id, Result&& result) const
    {
        ...
    }
};
```
where the parameters passed to the overloaded function operator `operator()` are:

- the `id` is the global id where the final result of the asynchronous chain of operations should be sent to (in most cases this is the id of the `hpx::future` returned from the initial call to `hpx::async_continue`. Any custom continuation function should make sure this `id` is forwarded to the last operation in the chain.
- the `result` is the result value of the current operation in the asynchronous execution chain. This value needs to be forwarded to the next operation.

**Note:** All of those operations are implemented by the predefined continuation function object which is returned from `hpx::make_continuation`. Any (custom) function object used as a continuation should conform to the same interface.

### Action error handling

Like in any other asynchronous invocation scheme it is important to be able to handle error conditions occurring while the asynchronous (and possibly remote) operation is executed. In HPX all error handling is based on standard C++ exception handling. Any exception thrown during the execution of an asynchronous operation will be transferred back to the original invocation `locality`, where it is rethrown during synchronization with the calling thread.

**Important:** Exceptions thrown during asynchronous execution can be transferred back to the invoking thread only for the synchronous and the asynchronous case with synchronization. Like with any other unhandled exception, any exception thrown during the execution of an asynchronous action without synchronization will result in calling `hpx::terminate` causing the running application to exit immediately.

**Note:** Even if error handling internally relies on exceptions, most of the API functions exposed by HPX can be used without throwing an exception. Please see Working with exceptions for more information.

As an example, we will assume that the following remote function will be executed:

```cpp
namespace app
{
    void some_function_with_error(int arg)
    {
        if (arg < 0) {
            HPX_THROW_EXCEPTION(bad_parameter, "some_function_with_error",
                                "some really bad error happened");
        }
        // do something else...
    }
}
```

// This will define the action type 'some_error_action' which represents
// the function 'app::some_function_with_error'.
HPX_PLAIN_ACTION(app::some_function_with_error, some_error_action);

The use of `HPX_THROW_EXCEPTION` to report the error encapsulates the creation of a `hpx::exception` which is initialized with the error code `hpx::bad_parameter`. Additionally it carries the passed strings, the information
about the file name, line number, and call stack of the point the exception was thrown from.

We invoke this action using the synchronous syntax as described before:

```cpp
// note: wrapped function will throw hpx::exception
some_error_action act; // define an instance of some_error_action
try {
    act(hpx::find_here(), -3); // exception will be rethrown from here
} catch (hpx::exception const & e) {
    // prints: 'some really bad error happened: HPX(bad parameter)'
    cout << e.what();
}
```

If this action is invoked asynchronously with synchronization, the exception is propagated to the waiting thread as well and is re-thrown from the future’s function `get()`:

```cpp
// note: wrapped function will throw hpx::exception
some_error_action act; // define an instance of some_error_action
hpx::future<void> f = hpx::async(act, hpx::find_here(), -3);
try {
    f.get(); // exception will be rethrown from here
} catch (hpx::exception const & e) {
    // prints: 'some really bad error happened: HPX(bad parameter)'
    cout << e.what();
}
```

For more information about error handling please refer to the section *Working with exceptions*. There we also explain how to handle error conditions without having to rely on exception.

### Writing components

A component in *HPX* is a C++ class which can be created remotely and for which its member functions can be invoked remotely as well. The following sections highlight how components can be defined, created, and used.

#### Defining components

In order for a C++ class type to be managed remotely in *HPX*, the type must be derived from the `hpx::components::component_base` template type. We call such C++ class types ‘components’.

Note that the component type itself is passed as a template argument to the base class:

```cpp
// header file some_componenthpp

#include <hpx/include/components.hpp>

namespace app {
    // Define a new component type 'some_component'
    struct some_component :
        hpx::components::component_base<some_component>
    {
        // This member function is has to be invoked remotely
        int some_member_function(std::string const & s)
        { return boost::lexical_cast<int>(s); }
    }
}
```

(continues on next page)
// This will define the action type 'some_member_action' which
// represents the member function 'some_member_function' of the
// object type 'some_component'.
HPX_DEFINE_COMPONENT_ACTION(some_component, some_member_function, some_member_
→action);
});

// This will generate the necessary boiler-plate code for the action allowing
// it to be invoked remotely. This declaration macro has to be placed in the
// header file defining the component itself.

// Note: The second argument to the macro below has to be systemwide-unique
// C++ identifiers
//
HPX_REGISTER_ACTION_DECLARATION(app::some_component::some_member_action, some_
→component_some_action);

There is more boiler plate code which has to be placed into a source file in order for the component to be usable. Every
component type is required to have macros placed into its source file, one for each component type and one macro for
each of the actions defined by the component type.

For instance:

// source file some_component.cpp
#include "some_component.hpp"

// The following code generates all necessary boiler plate to enable the
// remote creation of 'app::some_component' instances with 'hpx::new_<>()'
//
using some_component = app::some_component;
using some_component_type = hpx::components::component<some_component>;

// Please note that the second argument to this macro must be a
// (system-wide) unique C++-style identifier (without any namespaces)
//
HPX_REGISTER_COMPONENT(some_component_type, some_component);

// The parameters for this macro have to be the same as used in the corresponding
// HPX_REGISTER_ACTION_DECLARATION() macro invocation in the corresponding
// header file.

// Please note that the second argument to this macro must be a
// (system-wide) unique C++-style identifier (without any namespaces)
//
HPX_REGISTER_ACTION(app::some_component::some_member_action, some_component_some_
→action);

Defining client side representation classes

Often it is very convenient to define a separate type for a component which can be used on the client side (from
where the component is instantiated and used). This step might seem as unnecessary duplicating code, however it
significantly increases the type safety of the code.

A possible implementation of such a client side representation for the component described in the previous section
could look like:

```cpp
#include <hpx/include/components.hpp>

namespace app
{
    // Define a client side representation type for the component type
    // 'some_component' defined in the previous section.
    //
    // struct some_component_client
    //     : hpx::components::client_base<some_component_client, some_component>
    //     {
    //     using base_type = hpx::components::client_base<
    //         some_component_client, some_component>;
    //
    //     some_component_client(hpx::future<hpx::id_type> && id)
    //         : base_type(std::move(id))
    //     {}
    //
    //     hpx::future<int> some_member_function(std::string const & s)
    //         {
    //         some_component::some_member_action act;
    //         return hpx::async(act, get_id(), s);
    //     }
    // }
}
```

A client side object stores the global id of the component instance it represents. This global id is accessible by calling the function `client_base::get_id()`. The special constructor which is provided in the example allows to create this client side object directly using the API function `hpx::new_`.

**Creating component instances**

Instances of defined component types can be created in two different ways. If the component to create has a defined client side representation type, then this can be used, otherwise use the server type.

The following examples assume that `some_component_type` is the type of the server side implementation of the component to create. All additional arguments (see , . . . notation below) are passed through to the corresponding constructor calls of those objects:

```cpp
// create one instance on the given locality
hpx::id_type here = hpx::find_here();
hpx::future<hpx::id_type> f =
    hpx::new_<some_component_type>(here, ...);

// create one instance using the given distribution
// policy (here: hpx::colocating_distribution_policy)
// hpx::id_type here = hpx::find_here();
hpx::future<hpx::id_type> f =
    hpx::new_<some_component_type>(hpx::colocated(here), ...);

// create multiple instances on the given locality
hpx::id_type here = find_here();
hpx::future<std::vector<hpx::id_type>> f =
    hpx::new_<some_component_type[]>(here, num, ...);

// create multiple instances using the given distribution
// policy (here: hpx::binpacking_distribution_policy)
```

(continues on next page)
The examples below demonstrate the use of the same API functions for creating client side representation objects (instead of just plain ids). These examples assume that client_type is the type of the client side representation of the component type to create. As above, all additional arguments (see \ldots notation below) are passed through to the corresponding constructor calls of the server side implementation objects corresponding to the client_type:

```cpp
hpx::id_type here = hpx::find_here();
client_type c = hpx::new_<client_type>(here, ...);
```

```cpp
// create one instance on the given locality
hpx::id_type here = hpx::find_here();
client_type c = hpx::new_<client_type>(here, ...);
```

```cpp
// create one instance using the given distribution
// policy (here: hpx::colocating_distribution_policy)
hpx::id_type here = hpx::find_here();
client_type c = hpx::new_<client_type>(hpx::colocated(here), ...);
```

```cpp
// create multiple instances on the given locality
hpx::id_type here = hpx::find_here();
auto f = hpx::new_<client_type[]>(here, num, ...);
```

```cpp
// create multiple instances using the given distribution
// policy (here: hpx::binpacking_distribution_policy)
auto f = hpx::new_<client_type[]>(
    hpx::binpacking(hpx::find_all_localities()), num, ...);
```

### Using component instances

#### Segmented containers

In parallel programming, there is now a plethora of solutions aimed at implementing “partially contiguous” or segmented data structures, whether on shared memory systems or distributed memory systems. HPX implements such structures by drawing inspiration from Standard C++ containers.

#### Using segmented containers

A segmented container is a template class that is described in the namespace hpx. All segmented containers are very similar semantically to their sequential counterpart (defined in namespace std but with an additional template parameter named DistPolicy). The distribution policy is an optional parameter that is passed last to the segmented container constructor (after the container size when no default value is given, after the default value if not). The distribution policy describes the manner in which a container is segmented and the placement of each segment among the available runtime localities.

However, only a part of the std container member functions were reimplemented:

- (constructor), (destructor), operator=
- operator[
- begin, cbegin, end, cend
- size

An example of how to use the partitioned_vector container would be:
An example of how to use the `partitioned_vector` container with distribution policies would be:

```cpp
#include <hpx/include/partitioned_vector.hpp>
#include <hpx/runtime/find_localities.hpp>

HPX_REGISTER_PARTITIONED_VECTOR(double);

std::size_t num_segments = 10;
std::vector<hpx::id_type> locs = hpx::find_all_localities()

auto layout = hpx::container_layout(num_segments, locs);

hpx::partitioned_vector<double> va(50, layout);
```

By definition, a segmented container must be accessible from any thread although its construction is synchronous only for the thread who has called its constructor. To overcome this problem, it is possible to assign a symbolic name to the segmented container:

```cpp
#include <hpx/include/partitioned_vector.hpp>

HPX_REGISTER_PARTITIONED_VECTOR(double);

hpx::future<void> fserver = hpx::async(
    []()
    { hpx::partitioned_vector<double> v(50);

      // Register the 'partitioned_vector' with the name "some_name"
      //
      v.register_as("some_name");

      /* Do some code */
    });
```
Segmented containers

HPX provides the following segmented containers:

Table 2.25: Sequence containers

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>In header</th>
<th>Class page at cppreference.com</th>
</tr>
</thead>
<tbody>
<tr>
<td>hpx::partitioned_vector</td>
<td>Dynamic segmented contiguous array.</td>
<td>&lt;hpx/include/partitioned_vector.hpp&gt;</td>
<td>vector204</td>
</tr>
</tbody>
</table>

Table 2.26: Unordered associative containers

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>In header</th>
<th>Class page at cppreference.com</th>
</tr>
</thead>
<tbody>
<tr>
<td>hpx::unordered_map</td>
<td>Segmented collection of key-value pairs, hashed by keys, keys are unique.</td>
<td>&lt;hpx/include/unordered_map.hpp&gt;</td>
<td>unordered_map205</td>
</tr>
</tbody>
</table>

Segmented iterators and segmented iterator traits

The basic iterator used in the STL library is only suitable for one-dimensional structures. The iterators we use in HPX must adapt to the segmented format of our containers. Our iterators are then able to know when incrementing themselves if the next element of type T is in the same data segment or in another segment. In this second case, the iterator will automatically point to the beginning of the next segment.

Note: Note that the dereference operation operator * does not directly return a reference of type T& but an intermediate object wrapping this reference. When this object is used as an l-value, a remote write operation is performed; When this object is used as an r-value, implicit conversion to T type will take care of performing remote read operation.

It is sometimes useful not only to iterate element by element, but also segment by segment, or simply get a local iterator in order to avoid additional construction costs at each de-referencing operations. To mitigate this need, the hpx::traits::segmented_iterator_traits are used.

204 http://en.cppreference.com/w/cpp/container/vector
205 http://en.cppreference.com/w/cpp/container/unordered_map
With `segmented_iterator_traits` users can uniformly get the iterators which specifically iterates over segments (by providing a segmented iterator as a parameter), or get the local begin/end iterators of the nearest local segment (by providing a per-segment iterator as a parameter):

```cpp
#include <hpx/include/partitioned_vector.hpp>

// The following code generates all necessary boiler plate to enable the
// remote creation of 'partitioned_vector' segments
//
HPX_REGISTER_PARTITIONED_VECTOR(double);

using iterator = hpx::partitioned_vector<T>::iterator;
using traits = hpx::traits::segmented_iterator_traits<iterator>;

hpx::partitioned_vector<T> v;
std::size_t count = 0;

auto seg_begin = traits::segment(v.begin());
auto seg_end = traits::segment(v.end());

// Iterate over segments
for (auto seg_it = seg_begin; seg_it != seg_end; ++seg_it)
{
    auto loc_begin = traits::begin(seg_it)
    auto loc_end = traits::end(seg_it);

    // Iterate over elements inside segments
    for (auto lit = loc_begin; lit != loc_end; ++lit, ++count)
    {
        *lit = count;
    }
}
```

Which is equivalent to:

```cpp
hpx::partitioned_vector<T> v;
std::size_t count = 0;

auto begin = v.begin();
auto end = v.end();

for (auto it = begin; it != end; ++it, ++count)
{
    *it = count;
}
```

### Using views

The use of multidimensional arrays is quite common in the numerical field whether to perform dense matrix operations or to process images. It exist many libraries which implement such object classes overloading their basic operators (e.g. `"+", 
-", +", (), etc.). However, such operation becomes more delicate when the underlying data layout is segmented or when it is mandatory to use optimized linear algebra subroutines (i.e. BLAS subroutines).

Our solution is thus to relax the level of abstraction by allowing the user to work not directly on n-dimensional data, but on “n-dimensional collections of 1-D arrays”. The use of well-accepted techniques on contiguous data is thus preserved at the segment level, and the composability of the segments is made possible thanks to multidimensional array-inspired access mode.
Preface: Why SPMD?

Although HPX refutes by design this programming model, the locality plays a dominant role when it comes to implement vectorized code. To maximize local computations and avoid unneeded data transfers, a parallel section (or Single Programming Multiple Data section) is required. Because the use of global variables is prohibited, this parallel section is created via the RAII idiom.

To define a parallel section, simply write an action taking a spmd_block variable as a first parameter:

```cpp
#include <hpx/collectives/spmd_block.hpp>

void bulk_function(hpx::lcos::spmd_block block /* , arg0, arg1, ... */) {
    // Parallel section
    /* Do some code */
}
HPX_plain_action(bulk_function, bulk_action);
```

**Note:** In the following paragraphs, we will use the term “image” several times. An image is defined as a lightweight process whose entry point is a function provided by the user. It’s an “image of the function”.

The spmd_block class contains the following methods:
- [def Team information] get_num_images, this_image, images_per_locality
- [def Control statements] sync_all, sync_images

Here is a sample code summarizing the features offered by the spmd_block class:

```cpp
#include <hpx/collectives/spmd_block.hpp>

void bulk_function(hpx::lcos::spmd_block block /* , arg0, arg1, ... */) {
    std::size_t num_images = block.get_num_images();
    std::size_t this_image = block.this_image();
    std::size_t images_per_locality = block.images_per_locality();

    /* Do some code */

    // Synchronize all images in the team
    block.sync_all();

    /* Do some code */

    // Synchronize image 0 and image 1
    block.sync_images(0, 1);

    /* Do some code */

    std::vector<std::size_t> vec_images = {2, 3, 4};

    // Synchronize images 2, 3 and 4
    block.sync_images(vec_images);

    // Alternative call to synchronize images 2, 3 and 4
    block.sync_images(vec_images.begin(), vec_images.end());
}
```

(continues on next page)
/* Do some code */

// Non-blocking version of sync_all()
hx::future<void> event =
    block.sync_all(hpx::launch::async);

// Callback waiting for 'event' to be ready before being scheduled
hx::future<void> cb =
    event.then(
        [](hx::future<void>)
        {
            /* Do some code */
        });

// Finally wait for the execution tree to be finished
cb.get();

HPX_PLAIN_ACTION(bulk_test_function, bulk_test_action);

Then, in order to invoke the parallel section, call the function define_spmd_block specifying an arbitrary symbolic name and indicating the number of images per locality to create:

```cpp
void bulk_function(hpx::lcos::spmd_block block, /* , arg0, arg1, ... */)
{
    /* std::size_t arg0, arg1, ... */
    bulk_action act;
    std::size_t images_per_locality = 4;

    // Instantiate the parallel section
    hpx::lcos::define_spmd_block(
        "some_name", images_per_locality, std::move(act) /*, arg0, arg1, ... */);
    return 0;
}
```

**Note:** In principle, the user should never call the `spmd_block` constructor. The `define_spmd_block` function is responsible of instantiating `spmd_block` objects and broadcasting them to each created image.

### SPMD multidimensional views

Some classes are defined as “container views” when the purpose is to observe and/or modify the values of a container using another perspective than the one that characterizes the container. For example, the values of an `std::vector` object can be accessed via the expression `[i]`. Container views can be used, for example, when it is desired for those values to be “viewed” as a 2D matrix that would have been flattened in a `std::vector`. The values would be
possibly accessible via the expression $v(v(i,j))$ which would call internally the expression $v[k]$.

By default, the `partitioned_vector` class integrates 1-D views of its segments:

```cpp
#include <hpx/include/partitioned_vector.hpp>

// The following code generates all necessary boiler plate to enable the
// remote creation of 'partitioned_vector' segments
HPX_REGISTER_PARTITIONED_VECTOR(double);

using iterator = hpx::partitioned_vector<double>::iterator;
using traits = hpx::traits::segmented_iterator_traits<iterator>;

hpx::partitioned_vector<double> v;

// Create a 1-D view of the vector of segments
auto vv = traits::segment(v.begin());

// Access segment i
std::vector<double> v = vv[i];
```

Our views are called “multidimensional” in the sense that they generalize to N dimensions the purpose of `segmented_iterator_traits::segment()` in the 1-D case. Note that in a parallel section, the 2-D expression $a(i,j) = b(i,j)$ is quite confusing because without convention, each of the images invoked will race to execute the statement. For this reason, our views are not only multidimensional but also “spmd-aware”.

**Note:** SPMD-awareness: The convention is simple. If an assignment statement contains a view subscript as an l-value, it is only and only the image holding the r-value who is evaluating the statement. (In MPI sense, it is called a Put operation).

### Subscript-based operations

Here are some examples of using subscripts in the 2-D view case:

```cpp
#include <hpx/components/containers/partitioned_vector/partitioned_vector_view.hpp>
#include <hpx/include/partitioned_vector.hpp>

// The following code generates all necessary boiler plate to enable the
// remote creation of 'partitioned_vector' segments
HPX_REGISTER_PARTITIONED_VECTOR(double);

using Vec = hpx::partitioned_vector<double>;
using View_2D = hpx::partitioned_vector_view<double,2>;

/* Do some code */

Vec v;

// Parallel section (suppose 'block' an spmd_block instance)
{
    std::size_t height, width;
    // Instantiate the view
    View_2D vv(block, v.begin(), v.end(), {height,width});
}
```

(continues on next page)
// The l-value is a view subscript, the image that owns vv(1,0) evaluates the assignment.
vv(0,1) = vv(1,0);

// The l-value is a view subscript, the image that owns the r-value (result of expression 'std::vector<double>(4,1.0)') evaluates the assignment: oops! race between all participating images.
vv(2,3) = std::vector<double>(4,1.0);

**Iterator-based operations**

Here are some examples of using iterators in the 3-D view case:

```cpp
#include <hpx/components/containers/partitioned_vector/partitioned_vector_view.hpp>
#include <hpx/include/partitioned_vector.hpp>

// The following code generates all necessary boiler plate to enable the remote creation of 'partitioned_vector' segments
HPX_REGISTER_PARTITIONED_VECTOR(int);

using Vec = hpx::partitioned_vector<int>;
using View_3D = hpx::partitioned_vector_view<int,3>;

/* Do some code */

Vec v1, v2;

// Parallel section (suppose 'block' an spmd_block instance)
{
    std::size_t size_x, size_y, size_z;

    // Instantiate the views
    View_3D vv1(block, v1.begin(), v1.end(), {size_x,size_y,size_z});
    View_3D vv2(block, v2.begin(), v2.end(), {size_x,size_y,size_z});

    // Save previous segments covered by vv1 into segments covered by vv2
    auto vv2_it = vv2.begin();
    auto vv1_it = vv1.cbegin();
    for(; vv2_it != vv2.end(); vv2_it++, vv1_it++)
    {
        // It's a Put operation
        *vv2_it = *vv1_it;
    }

    // Ensure that all images have performed their Put operations
    block.sync_all();

    // Ensure that only one image is putting updated data into the different segments covered by vv1
    if(block.this_image() == 0)
    {
        int idx = 0;
    }
}
```

(continues on next page)
// Update all the segments covered by vv1
for (auto i = vv1.begin(); i != vv1.end(); i++)
{
    // It's a Put operation
    *i = std::vector<float>(elt_size, idx++);
}
}
}

Here is an example that shows how to iterate only over segments owned by the current image:

```cpp
#include <hpx/components/containers/partitioned_vector/partitioned_vector_view.hpp>
#include <hpx/components/containers/partitioned_vector/partitioned_vector_local_view.hpp>
#include <hpx/include/partitioned_vector.hpp>

// The following code generates all necessary boiler plate to enable the
// remote creation of 'partitioned_vector' segments
//
HPX_REGISTER_PARTITIONED_VECTOR(float);

using Vec = hpx::partitioned_vector<float>;
using View_1D = hpx::partitioned_vector_view<float, 1>;

/* Do some code */
Vec v;

// Parallel section (suppose 'block' an spmd_block instance)
{
    std::size_t num_segments;

    // Instanciate the view
    View_1D vv(block, v.begin(), v.end(), {num_segments});

    // Instanciate the local view from the view
    auto local_vv = hpx::local_view(vv);

    for ( auto i = localvv.begin(); i != localvv.end(); i++ )
    {
        std::vector<float> & segment = *i;
        /* Do some code */
    }
}

Instanciating sub-views

It is possible to construct views from other views: we call it sub-views. The constraint nevertheless for the subviews is to retain the dimension and the value type of the input view. Here is an example showing how to create a sub-view:

```cpp
#include <hpx/components/containers/partitioned_vector/partitioned_vector_view.hpp>
#include <hpx/include/partitioned_vector.hpp>
```
(continues on next page)
// The following code generates all necessary boiler plate to enable the
// remote creation of 'partitioned_vector' segments
//
HPX_REGISTER_PARTITIONED_VECTOR(float);

using Vec = hpx::partitioned_vector<float>;
using View_2D = hpx::partitioned_vector_view<float,2>;

/* Do some code */

Vec v;

// Parallel section (suppose 'block' an spmd_block instance)
{
    std::size_t N = 20;
    std::size_t tilesize = 5;

    // Instanciate the view
    View_2D vv(block, v.begin(), v.end(), {N,N});

    // Instanciate the subview
    View_2D svv(
        block, &vv(tilesize,0), &vv(2*tilesize-1,tilesize-1), {tilesize,tilesize}, {N,N});

    if(block.this_image() == 0)
    {
        // Equivalent to 'vv(tilesize,0) = 2.0f'
        svv(0,0) = 2.0f;

        // Equivalent to 'vv(2*tilesize-1,tilesize-1) = 3.0f'
        svv(tilesize-1,tilesize-1) = 3.0f;
    }
}

Note: The last parameter of the subview constructor is the size of the original view. If one would like to create a
subview of the subview and so on, this parameter should stay unchanged. (N, N) for the above example.

C++ co-arrays

Fortran has extended its scalar element indexing approach to reference each segment of a distributed array. In this
extension, a segment is attributed a ?co-index? and lives in a specific locality. A co-index provides the application
with enough information to retrieve the corresponding data reference. In C++, containers present themselves as a
?smarter? alternative of Fortran arrays but there are still no corresponding standardized features similar to the Fortran
co-indexing approach. We present here an implementation of such features in HPX.

Preface: co-array, a segmented container tied to a SPMD multidimensional views

As mentioned before, a co-array is a distributed array whose segments are accessible through an array-inspired access
mode. We have previously seen that it is possible to reproduce such access mode using the concept of views. Nev-
ertheless, the user must pre-create a segmented container to instanciate this view. We illustrate below how a single
constructor call can perform those two operations:
#include <hpx/components/containers/coarray/coarray.hpp>
#include <hpx/collectives/spmd_block.hpp>

// The following code generates all necessary boiler plate to enable the 
// co-creation of 'coarray'
HPX_REGISTER_COARRAY(double);

// Parallel section (suppose 'block' an spmd_block instance)
{
  using hpx::container::placeholders::_;
  std::size_t height=32, width=4, segment_size=10;
  hpx::coarray<double,3> a(block, "a", {height,width,_}, segment_size);
  /* Do some code */
}

Unlike segmented containers, a co-array object can only be instantiated within a parallel section. Here is the description of the parameters to provide to the coarray constructor:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>block</td>
<td>Reference to a spmd_block object</td>
</tr>
<tr>
<td>&quot;a&quot;</td>
<td>Symbolic name of type std::string</td>
</tr>
<tr>
<td>{height, width, _}</td>
<td>Dimensions of the coarray object</td>
</tr>
<tr>
<td>segment_size</td>
<td>Size of a co-indexed element (i.e. size of the object referenced by the expression a(i, j,k))</td>
</tr>
</tbody>
</table>

Note that the “last dimension size” cannot be set by the user. It only accepts the constexpr variable hpx::container::placeholders::_. This size, which is considered private, is equal to the number of current images (value returned by block.get_num_images()).

Note: An important constraint to remember about coarray objects is that all segments sharing the same “last dimension index” are located in the same image.

## Using co-arrays

The member functions owned by the coarray objects are exactly the same as those of spmd multidimensional views. These are:

- Subscript-based operations
- Iterator-based operations

However, one additional functionality is provided. Knowing that the element \(a(i, j, k)\) is in the memory of the \(k\)th image, the use of local subscripts is possible.

Note: For spmd multidimensional views, subscripts are only global as it still involves potential remote data transfers.

Here is an example of using local subscripts:


```cpp
#include <hpx/components/containers/coarray/coarray.hpp>
#include <hpx/collectives/spmd_block.hpp>

// The following code generates all necessary boiler plate to enable the
// co-creation of 'coarray'
//
HPX_REGISTER_COARRAY(double);

// Parallel section (suppose 'block' an spmd_block instance)
{
  using hpx::container::placeholders::_;

  std::size_t height = 32, width = 4, segment_size = 10;

  hpx::coarray<double, 3> a(block, "a", {height, width, _}, segment_size);

  double idx = block.this_image() * height * width;

  for (std::size_t j = 0; j < width; j++)
    for (std::size_t i = 0; i < height; i++)
      { // Local write operation performed via the use of local subscript
        a(i, j, _) = std::vector<double>(elt_size, idx);
        idx++;
      }

  block.sync_all();
}
```

Note: When the “last dimension index” of a subscript is equal to `hpx::container::placeholders::_`, local subscript (and not global subscript) is used. It is equivalent to a global subscript used with a “last dimension index” equal to the value returned by `block.this_image()`.

---

### 2.5.8 Running on batch systems

This section walks you through launching HPX applications on various batch systems.

#### How to use HPX applications with PBS

Most HPX applications are executed on parallel computers. These platforms typically provide integrated job management services that facilitate the allocation of computing resources for each parallel program. HPX includes support for one of the most common job management systems, the Portable Batch System (PBS).

All PBS jobs require a script to specify the resource requirements and other parameters associated with a parallel job. The PBS script is basically a shell script with PBS directives placed within commented sections at the beginning of the file. The remaining (not commented-out) portions of the file executes just like any other regular shell script. While the description of all available PBS options is outside the scope of this tutorial (the interested reader may refer to in-depth documentation[^206] for more information), below is a minimal example to illustrate the approach. The following test application will use the multithreaded `hello_world_distributed` program, explained in the section Remote execution with actions: Hello world.

```bash
#!/bin/bash
# (continues on next page)
```

[^206]: http://www.clusterresources.com/torquedocs21/
#PBS -l nodes=2:ppn=4
APP_PATH=/packages/hpx/bin/hello_world_distributed
APP_OPTIONS=

pbsdsh -u $APP_PATH $APP_OPTIONS --hpx:nodes=`cat $PBS_NODEFILE`

**Caution:** If the first application specific argument (inside $APP_OPTIONS) is a non-option (i.e., does not start with a `-` or a `--`), then the argument has to be placed before the option `--hpx:nodes`, which, in this case, should be the last option on the command line.

Alternatively, use the option `--hpx:endnodes` to explicitly mark the end of the list of node names:

```
pbsdsh -u $APP_PATH --hpx:nodes`cat $PBS_NODEFILE` --hpx:endnodes $APP_OPTIONS
```

The #PBS -l nodes=2:ppn=4 directive will cause two compute nodes to be allocated for the application, as specified in the option nodes. Each of the nodes will dedicate four cores to the program, as per the option ppn, short for “processors per node” (PBS does not distinguish between processors and cores). Note that requesting more cores per node than physically available is pointless and may prevent PBS from accepting the script.

On newer PBS versions the PBS command syntax might be different. For instance, the PBS script above would look like:

```
#!/bin/bash
#
#PBS -l select=2:ncpus=4
APP_PATH=/packages/hpx/bin/hello_world_distributed
APP_OPTIONS=

pbsdsh -u $APP_PATH $APP_OPTIONS --hpx:nodes=`cat $PBS_NODEFILE`
```

APP_PATH and APP_OPTIONS are shell variables that respectively specify the correct path to the executable (hello_world_distributed in this case) and the command line options. Since the hello_world_distributed application doesn’t need any command line options, APP_OPTIONS has been left empty. Unlike in other execution environments, there is no need to use the `--hpx:threads` option to indicate the required number of OS threads per node; the HPX library will derive this parameter automatically from PBS.

Finally, pbsdsh is a PBS command that starts tasks to the resources allocated to the current job. It is recommended to leave this line as shown and modify only the PBS options and shell variables as needed for a specific application.

**Important:** A script invoked by pbsdsh starts in a very basic environment: the user’s $HOME directory is defined and is the current directory, the LANG variable is set to C and the PATH is set to the basic /usr/local/bin:/usr/bin:/bin as defined in a system-wide file pbs_environment. Nothing that would normally be set up by a system shell profile or user shell profile is defined, unlike the environment for the main job script.

Another choice is for the pbsdsh command in your main job script to invoke your program via a shell, like sh or bash, so that it gives an initialized environment for each instance. Users can create a small script runme.sh, which is used to invoke the program:

```
#!/bin/bash
# Small script which invokes the program based on what was passed on its
```

(continues on next page)
# command line.
#
# This script is executed by the bash shell which will initialize all
# environment variables as usual.
$@

Now, the script is invoked using the pbsdsh tool:

```bash
#!/bin/bash
#
#PBS -l nodes=2:ppn=4

APP_PATH=~/packages/hpx/bin/hello_world_distributed
APP_OPTIONS=

pbsdsh -u runme.sh $APP_PATH $APP_OPTIONS --hpx:nodes=`cat $PBS_NODEFILE`
```

All that remains now is submitting the job to the queuing system. Assuming that the contents of the PBS script were saved in the file `pbs_hello_world.sh` in the current directory, this is accomplished by typing:

```bash
qsub ./pbs_hello_world_pbs.sh
```

If the job is accepted, qsub will print out the assigned job ID, which may look like:

```bash
$ 42.supercomputer.some.university.edu
```

To check the status of your job, issue the following command:

```bash
qstat 42.supercomputer.some.university.edu
```

and look for a single-letter job status symbol. The common cases include:

- **Q** - signifies that the job is queued and awaiting its turn to be executed.
- **R** - indicates that the job is currently running.
- **C** - means that the job has completed.

The example qstat output below shows a job waiting for execution resources to become available:

<table>
<thead>
<tr>
<th>Job id</th>
<th>Name</th>
<th>User</th>
<th>Time</th>
<th>Use</th>
<th>S</th>
<th>Queue</th>
</tr>
</thead>
<tbody>
<tr>
<td>42.supercomputer</td>
<td>...ello_world.sh joe_user</td>
<td></td>
<td>0</td>
<td></td>
<td>Q</td>
<td>batch</td>
</tr>
</tbody>
</table>

After the job completes, PBS will place two files, `pbs_hello_world.sh.042` and `pbs_hello_world.sh.042.e`, in the directory where the job was submitted. The first contains the standard output and the second contains the standard error from all the nodes on which the application executed. In our example, the error output file should be empty and the standard output file should contain something similar to:

```
hello world from OS-thread 3 on locality 0
hello world from OS-thread 2 on locality 0
hello world from OS-thread 1 on locality 1
hello world from OS-thread 0 on locality 0
hello world from OS-thread 3 on locality 1
hello world from OS-thread 2 on locality 1
hello world from OS-thread 1 on locality 0
hello world from OS-thread 0 on locality 1
```
Congratulations! You have just run your first distributed HPX application!

**How to use HPX applications with SLURM**

Just like PBS (described in section *How to use HPX applications with PBS*), SLURM is a job management system which is widely used on large supercomputing systems. Any HPX application can easily be run using SLURM. This section describes how this can be done.

The easiest way to run an HPX application using SLURM is to utilize the command line tool srun, which interacts with the SLURM batch scheduling system:

```
slurm -p <partition> -N <number-of-nodes> hpx-application <application-arguments>
```

Here, `<partition>` is one of the node partitions existing on the target machine (consult the machine’s documentation to get a list of existing partitions) and `<number-of-nodes>` is the number of compute nodes that should be used. By default, the HPX application is started with one *locality* per node and uses all available cores on a node. You can change the number of localities started per node (for example, to account for NUMA effects) by specifying the `-n` option of srun. The number of cores per *locality* can be set by `-c`. The `<application-arguments>` are any application specific arguments that need to be passed on to the application.

**Note:** There is no need to use any of the HPX command line options related to the number of localities, number of threads, or related to networking ports. All of this information is automatically extracted from the SLURM environment by the HPX startup code.

**Important:** The srun documentation explicitly states: “If `-c` is specified without `-n`, as many tasks will be allocated per node as possible while satisfying the `-c` restriction. For instance on a cluster with 8 CPUs per node, a job request for 4 nodes and 3 CPUs per task may be allocated 3 or 6 CPUs per node (1 or 2 tasks per node) depending upon resource consumption by other jobs.” For this reason, it’s recommended to always specify `-n <number-of-instances>`, even if `<number-of-instances>` is equal to one (1).

**Interactive shells**

To get an interactive development shell on one of the nodes, users can issue the following command:

```
slurm -p <node-type> -N <number-of-nodes> --pty /bin/bash -l
```

After the shell has been opened, users can run their HPX application. By default, it uses all available cores. Note that if you requested one node, you don’t need to do slurm again. However, if you requested more than one node, and want to run your distributed application, you can use slurm again to start up the distributed HPX application. It will use the resources that have been requested for the interactive shell.

**Scheduling batch jobs**

The above mentioned method of running HPX applications is fine for development purposes. The disadvantage that comes with slurm is that it only returns once the application is finished. This might not be appropriate for longer-running applications (for example, benchmarks or larger scale simulations). In order to cope with that limitation, users can use the sbatch command.

The sbatch command expects a script that it can run once the requested resources are available. In order to request resources, users need to add `#SBATCH` comments in their script or provide the necessary parameters to sbatch directly. The parameters are the same as with run. The commands you need to execute are the same you would need to start your application as if you were in an interactive shell.
2.5.9 Debugging HPX applications

Using a debugger with HPX applications

Using a debugger such as gdb with HPX applications is no problem. However, there are some things to keep in mind to make the experience somewhat more productive.

Call stacks in HPX can often be quite unwieldy as the library is heavily templated and the call stacks can be very deep. For this reason it is sometimes a good idea to compile HPX in RelWithDebInfo mode, which applies some optimizations but keeps debugging symbols. This can often compress call stacks significantly. On the other hand, stepping through the code can also be more difficult because of statements being reordered and variables being optimized away. Also, note that because HPX implements user-space threads and context switching, call stacks may not always be complete in a debugger.

HPX launches not only worker threads but also a few helper threads. The first thread is the main thread, which typically does no work in an HPX application, except at startup and shutdown. If using the default settings, HPX will spawn six additional threads (used for service thread pools). The first worker thread is usually the eighth thread, and most user codes will be run on these worker threads. The last thread is a helper thread used for HPX shutdown.

Finally, since HPX is a multi-threaded runtime, the following gdb options can be helpful:

```bash
set pagination off
set non-stop on
```

Non-stop mode allows users to have a single thread stop on a breakpoint without stopping all other threads as well.

Using sanitizers with HPX applications

**Warning:** Not all parts of HPX are sanitizer clean. This means that users may end up with false positives from HPX itself when using sanitizers for their applications.

To use sanitizers with HPX, turn on HPX_WITH_SANITIZERS and turn off HPX_WITH_STACK_OVERFLOW_DETECTION during CMake configuration. It’s recommended to also build Boost with the same sanitizers that will be used for HPX. The appropriate sanitizers can then be enabled using CMake by appending -fsanitize=address -fno-omit-frame-pointer to CMAKE_CXX_FLAGS and -fsanitize=address to CMAKE_EXE_LINKER_FLAGS. Replace address with the sanitizer that you want to use.

2.5.10 Optimizing HPX applications

Performance counters

Performance Counters in HPX are used to provide information as to how well the runtime system or an application is performing. The counter data can help determine system bottlenecks and fine-tune system and application performance. The HPX runtime system, its networking, and other layers provide counter data that an application can consume to provide users with information of how well the application is performing.

Applications can also use counter data to determine how much system resources to consume. For example, an application that transfers data over the network could consume counter data from a network switch to determine how much data to transfer without competing for network bandwidth with other network traffic. The application could use the counter data to adjust its transfer rate as the bandwidth usage from other network traffic increases or decreases.

Performance Counters are HPX parallel processes which expose a predefined interface. HPX exposes special API functions that allow one to create, manage, read the counter data, and release instances of Performance Counters. Performance Counter instances are accessed by name, and these names have a predefined structure which is described in the section *Performance counter names*. The advantage of this is that any Performance Counter can be accessed remotely (from a different locality) or locally (from the same locality). Moreover, since all counters expose their data
using the same API, any code consuming counter data can be utilized to access arbitrary system information with 
minimal effort.

Counter data may be accessed in real time. More information about how to consume counter data can be found in the 
section Consuming performance counter data.

All HPX applications provide command line options related to performance counters, such as the ability to list available 
counter types, or periodically query specific counters to be printed to the screen or save them in a file. For more 
information, please refer to the section HPX Command Line Options.

Performance counter names

All Performance Counter instances have a name uniquely identifying this instance. This name can be used to access 
the counter, retrieve all related meta data, and to query the counter data (as described in the section Consuming 
performance counter data). Counter names are strings with a predefined structure. The general form of a countername 
is:

\[ /\text{objectname}\{\text{full_instancename}\}/\text{countername}@\text{parameters} \]

where full_instancename could be either another (full) counter name or a string formatted as:

\[ \text{parentinstancename}\#\text{parentindex}/\text{instancename}\#\text{instanceindex} \]

Each separate part of a countername (e.g. objectname, countername parentinstancename, 
instancename, and parameters) should start with a letter ('a'...'z', 'A'...'Z') or an underscore character ('_'), optionally followed by letters, digits ('0'...'9'), hyphen ('-'), or underscore characters. Whitespace 
is not allowed inside a counter name. The characters '/' , '{' , '}' , '#' and '@' have a special meaning and are 
used to delimit the different parts of the counter name.

The parts parentinstanceindex and instanceindex are integers. If an index is not specified HPX will 
assume a default of -1.

Two simple examples

An instance for a well formed (and meaningful) simple counter name would be:

\[ /\text{threads}\{\text{locality\#0/total}\}/\text{count}/\text{cumulative} \]

This counter returns the current cumulative number of executed (retired) HPX-threads for the locality 0. The counter 
type of this counter is /threads/count/cumulative and the full instance name is locality#0/total. This counter type does not require an instanceindex or parameters to be specified.

In this case, the parentindex (the '0') designates the locality for which the counter instance is created. The 
counter will return the number of HPX-threads retired on that particular locality.

Another example for a well formed (aggregate) counter name is:

\[ /\text{statistics}\{/\text{threads}\{\text{locality\#0/total}\}/\text{count}/\text{cumulative}\}/\text{average}@500 \]

This counter takes the simple counter from the first example, samples its values every 500 milliseconds, and returns the 
average of the value samples whenever it is queried. The counter type of this counter is /statistics/average and 
the instance name is the full name of the counter for which the values have to be averaged. In this case, the 
parameters (the '500') specify the sampling interval for the averaging to take place (in milliseconds).

Performance counter types

Every Performance Counter belongs to a specific Performance Counter type which classifies the counters into groups 
of common semantics. The type of a counter is identified by the objectname and the countername parts of the
When an application starts HPX will register all available counter types on each of the localities. These counter types are held in a special Performance Counter registration database which can be later used to retrieve the meta data related to a counter type and to create counter instances based on a given counter instance name.

Performance counter instances

The full_instancename distinguishes different counter instances of the same counter type. The formatting of the full_instancename depends on the counter type. There are two types of counters: simple counters which usually generate the counter values based on direct measurements, and aggregate counters which take another counter and transform its values before generating their own counter values. An example for a simple counter is given above: counting retired HPX-threads. An aggregate counter is shown as an example above as well: calculating the average of the underlying counter values sampled at constant time intervals.

While simple counters use instance names formatted as parentinstancename#parentindex/instancename#instanceindex, most aggregate counters have the full counter name of the embedded counter as its instance name.

Not all simple counter types require specifying all 4 elements of a full counter instance name, some of the parts (parentinstancename, parentindex, instancename, and instanceindex) are optional for specific counters. Please refer to the documentation of a particular counter for more information about the formatting requirements for the name of this counter (see Existing HPX performance counters).

The parameters are used to pass additional information to a counter at creation time. They are optional and they fully depend on the concrete counter. Even if a specific counter type allows additional parameters to be given, those usually are not required as sensible defaults will be chosen. Please refer to the documentation of a particular counter for more information about what parameters are supported, how to specify them, and what default values are assumed (see also Existing HPX performance counters).

Every locality of an application exposes its own set of Performance Counter types and Performance Counter instances. The set of exposed counters is determined dynamically at application start based on the execution environment of the application. For instance, this set is influenced by the current hardware environment for the locality (such as whether the locality has access to accelerators), and the software environment of the application (such as the number of OS-threads used to execute HPX-threads).

Using wildcards in performance counter names

It is possible to use wildcard characters when specifying performance counter names. Performance counter names can contain 2 types of wildcard characters:

- Wildcard characters in the performance counter type
- Wildcard characters in the performance counter instance name

Wildcard character have a meaning which is very close to usual file name wildcard matching rules implemented by common shells (like bash).

<table>
<thead>
<tr>
<th>Wildcard</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>This wildcard character matches any number (zero or more) of arbitrary characters.</td>
</tr>
<tr>
<td>?</td>
<td>This wildcard character matches any single arbitrary character.</td>
</tr>
<tr>
<td>[...]</td>
<td>This wildcard character matches any single character from the list of specified within the square brackets.</td>
</tr>
</tbody>
</table>
Table 2.29: Wildcard characters in the performance counter instance name

<table>
<thead>
<tr>
<th>Wildcard</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>This wildcard character matches any <em>locality</em> or any thread, depending on whether it is used for <em>locality#</em> or <em>worker-thread#</em>. No other wildcards are allowed in counter instance names.</td>
</tr>
</tbody>
</table>

Consuming performance counter data

You can consume performance data using either the command line interface or via the *HPX* application or the *HPX* API. The command line interface is easier to use, but it is less flexible and does not allow one to adjust the behaviour of your application at runtime. The command line interface provides a convenience abstraction but simplified abstraction for querying and logging performance counter data for a set of performance counters.

Consuming performance counter data from the command line

*HPX* provides a set of predefined command line options for every application which uses *hpx::init* for its initialization. While there are much more command line options available (see *HPX Command Line Options*), the set of options related to Performance Counters allow one to list existing counters, query existing counters once at application termination or repeatedly after a constant time interval.

The following table summarizes the available command line options:

Table 2.30: *HPX* Command Line Options Related to Performance Counters

<table>
<thead>
<tr>
<th>Command line option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--hpx:print-counter</td>
<td>print the specified performance counter either repeatedly and/or at the times specified by --hpx:print-counter-at (see also option --hpx:print-counter-interval).</td>
</tr>
<tr>
<td>--hpx:print-counter-at</td>
<td>print the specified performance counter either repeatedly and/or at the times specified by --hpx:print-counter-at or reset the counter after the value is queried. (see also option --hpx:print-counter-interval).</td>
</tr>
<tr>
<td>--hpx:print-counter</td>
<td>print the performance counter(s) specified with --hpx:print-counter repeatedly after the time interval (specified in milliseconds) (default: 0 which means print once at shutdown).</td>
</tr>
<tr>
<td>--hpx:print-counter</td>
<td>print the performance counter(s) specified with --hpx:print-counter to the given file (default: console).</td>
</tr>
<tr>
<td>--hpx:list-counters</td>
<td>list the names of all registered performance counters.</td>
</tr>
<tr>
<td>--hpx:list-counter-infos</td>
<td>list the description of all registered performance counters.</td>
</tr>
<tr>
<td>--hpx:print-counter-shortname,full-countername</td>
<td>print the performance counter(s) specified with --hpx:print-counter possible formats in csv format with header or without any header (see option --hpx:no-csv-header), possible values: csv (prints counter values in CSV format with full names as header) csv-short (prints counter values in CSV format with shortnames provided with --hpx:print-counter-shortname,full-countername)</td>
</tr>
<tr>
<td>--hpx:print-counter</td>
<td>print the performance counter(s) specified with --hpx:print-counter and possible formats with --hpx:print-counter-format without header.</td>
</tr>
<tr>
<td>--hpx:print-counter-reset</td>
<td>print the performance counter(s) specified with --hpx:print-counter (or --hpx:print-counter-reset) at the given point in time, possible argument values: startup, shutdown (default), noshutdown.</td>
</tr>
<tr>
<td>--hpx:reset-counters</td>
<td>reset all performance counter(s) specified with --hpx:print-counter after they have been evaluated</td>
</tr>
</tbody>
</table>
While the options `--hpx:list-counters` and `--hpx:list-counter-infos` give a short listing of all available counters, the full documentation for those can be found in the section *Existing HPX performance counters*.

### A simple example

All of the commandline options mentioned above can be for instance tested using the `hello_world_distributed` example.

#### Listing all available counters

`hello_world_distributed --hpx:list-counters` yields:

```
List of available counter instances (replace * below with the appropriate sequence number)
-------------------------------------------------------------------------
/agas/count/allocate /agas/count/bind /agas/count/bind_gid
/agas/count/bind_name ... /threads{locality#*/allocator#*}/count/objects
/threads{locality#*/total}/count/stack-recycles
/threads{locality#*/total}/idle-rate
/threads{locality#*/worker-thread#*}/idle-rate
```

#### Providing more information about all available counters

`hello_world_distributed --hpx:list-counter-infos` yields:

```
Information about available counter instances (replace * below with the appropriate sequence number)
-------------------------------------------------------------------------
fullname: /agas/count/allocate helptext: returns the number of invocations of the AGAS service 'allocate' type: counter_raw version: 1.0.0
-------------------------------------------------------------------------
 fullname: /agas/count/bind helptext: returns the number of invocations of the AGAS service 'bind' type: counter_raw version: 1.0.0
-------------------------------------------------------------------------
 fullname: /agas/count/bind_gid helptext: returns the number of invocations of the AGAS service 'bind_gid' type: counter_raw version: 1.0.0
-------------------------------------------------------------------------
...
```

This command will not only list the counter names but also a short description of the data exposed by this counter.

**Note:** The list of available counters may differ depending on the concrete execution environment (hardware or software) of your application.

Requesting the counter data for one or more performance counters can be achieved by invoking `hello_world_distributed` with a list of counter names:

```
hello_world_distributed \
  --hpx:print-counter=/threads{locality#0/total}/count/cumulative \
  --hpx:print-counter=/agas{locality#0/total}/count/bind
```

which yields for instance:
The first line is the normal output generated by `hello_world_distributed` and has no relation to the counter data listed. The last two lines contain the counter data as gathered at application shutdown. These lines have 6 fields, the counter name, the sequence number of the counter invocation, the time stamp at which this information has been sampled, the unit of measure for the time stamp, the actual counter value, and an optional unit of measure for the counter value.

The actual counter value can be represented by a single number (for counters returning singular values) or a list of numbers separated by `:` (for counters returning an array of values, like for instance a histogram).

**Note:** The name of the performance counter will be enclosed in double quotes `"` if it contains one or more commas `,`.

Requesting to query the counter data once after a constant time interval with this command line:

```bash
hello_world_distributed \
--hpx:print-counter=/threads{locality#0/total}/count/cumulative \
--hpx:print-counter=/agas{locality#0/total}/count/bind \
--hpx:print-counter-interval=20
```

yields for instance (leaving off the actual console output of the `hello_world_distributed` example for brevity):

```
threads{locality#0/total}/count/cumulative,1,0.002409,[s],22
agas{locality#0/total}/count/bind,1,0.002542,[s],9
threads{locality#0/total}/count/cumulative,2,0.023002,[s],41
agas{locality#0/total}/count/bind,2,0.023557,[s],10
threads{locality#0/total}/count/cumulative,3,0.037514,[s],46
agas{locality#0/total}/count/bind,3,0.038679,[s],10
```

The command `--hpx:print-counter-destination=<file>` will redirect all counter data gathered to the specified file name, which avoids cluttering the console output of your application.

The command line option `--hpx:print-counter` supports using a limited set of wildcards for a (very limited) set of use cases. In particular, all occurrences of `#*` as in `locality#*` and in `worker-thread#*` will be automatically expanded to the proper set of performance counter names representing the actual environment for the executed program. For instance, if your program is utilizing 4 worker threads for the execution of HPX threads (see command line option `--hpx:threads`) the following command line

```bash
hello_world_distributed \
--hpx:threads=4 \
--hpx:print-counter=/threads{locality#0/worker-thread#*}/count/cumulative
```

will print the value of the performance counters monitoring each of the worker threads:

```
hello world from OS-thread 1 on locality 0
hello world from OS-thread 0 on locality 0
hello world from OS-thread 3 on locality 0
hello world from OS-thread 2 on locality 0
/threads{locality#0/worker-thread#0}/count/cumulative,1,0.0025214,[s],27
/threads{locality#0/worker-thread#1}/count/cumulative,1,0.0025453,[s],33
/threads{locality#0/worker-thread#2}/count/cumulative,1,0.0025683,[s],29
/threads{locality#0/worker-thread#3}/count/cumulative,1,0.0025904,[s],33
```
The command `--hpx:print-counter-format` takes values `csv` and `csv-short` to generate CSV formatted counter values with header.

With format as csv:

```plaintext
hello_world_distributed
   --hpx:threads=2
   --hpx:print-counter-format csv
   --hpx:print-counter /threads{locality#*/total}/count/cumulative
   --hpx:print-counter /threads{locality#*/total}/count/cumulative-phases
```

will print the values of performance counters in CSV format with full countername as header:

```
hello world from OS-thread 1 on locality 0
hello world from OS-thread 0 on locality 0
```

With format csv-short:

```plaintext
hello_world_distributed
   --hpx:threads 2
   --hpx:print-counter-format csv-short
   --hpx:print-counter cumulative,/threads{locality#*/total}/count/cumulative
   --hpx:print-counter phases,/threads{locality#*/total}/count/cumulative-phases
```

will print the values of performance counters in CSV format with short countername as header:

```
hello world from OS-thread 1 on locality 0
hello world from OS-thread 0 on locality 0
```

With format csv and csv-short when used with `--hpx:print-counter-interval`:

```plaintext
hello_world_distributed
   --hpx:threads 2
   --hpx:print-counter-format csv-short
   --hpx:print-counter cumulative,/threads{locality#*/total}/count/cumulative
   --hpx:print-counter phases,/threads{locality#*/total}/count/cumulative-phases
   --hpx:print-counter-interval 5
```

will print the header only once repeating the performance counter value(s) repeatedly:

```
cum,phases
25,42
```

The command `--hpx:no-csv-header` to be used with `--hpx:print-counter-format` to print performance counter values in CSV format without any header:

```plaintext
hello_world_distributed
   --hpx:threads 2
   --hpx:print-counter-format csv-short
   --hpx:print-counter cumulative,/threads{locality#*/total}/count/cumulative
```

(continues on next page)
will print:

```
hello world from OS-thread 1 on locality 0
hello world from OS-thread 0 on locality 0
37,91
```

### Consuming performance counter data using the HPX API

HPX provides an API allowing to discover performance counters and to retrieve the current value of any existing performance counter from any application.

#### Discover existing performance counters

#### Retrieve the current value of any performance counter

Performance counters are specialized HPX components. In order to retrieve a counter value, the performance counter needs to be instantiated. HPX exposes a client component object for this purpose:

```cpp
hpx::performance_counters::performance_counter counter(std::string const & name);
```

Instantiating an instance of this type will create the performance counter identified by the given name. Only the first invocation for any given counter name will create a new instance of that counter, all following invocations for a given counter name will reference the initially created instance. This ensures, that at any point in time there is always not more than one active instance of any of the existing performance counters.

In order to access the counter value (or invoking any of the other functionality related to a performance counter, like start, stop or reset) member functions of the created client component instance should be called:

```cpp
// print the current number of threads created on locality 0
hpx::cout << counter.get_value<int>().get() << hpx::endl;
```

For more information about the client component type see [classref hpx::performance_counters::performance_counter].

**Note:** In the above example `counter.get_value()` returns a future. In order to print the result we must append `.get()` to retrieve the value. You could write the above example like this for more clarity:

```cpp
// print the current number of threads created on locality 0
hpx::future<int> result = counter.get_value<int>();
hpx::cout << result.get() << hpx::endl;
```

### Providing performance counter data

HPX offers several ways by which you may provide your own data as a performance counter. This has the benefit of exposing additional, possibly application specific information using the existing Performance Counter framework, unifying the process of gathering data about your application.
An application that wants to provide counter data can implement a Performance Counter to provide the data. When a consumer queries performance data, the HPX runtime system calls the provider to collect the data. The runtime system uses an internal registry to determine which provider to call.

Generally, there are two ways of exposing your own Performance Counter data: a simple, function based way and a more complex, but more powerful way of implementing a full Performance Counter. Both alternatives are described in the following sections.

**Exposing performance counter data using a simple function**

The simplest way to expose arbitrary numeric data is to write a function which will then be called whenever a consumer queries this counter. Currently, this type of Performance Counter can only be used to expose integer values. The expected signature of this function is:

```cpp
std::int64_t some_performance_data(bool reset);
```

The argument `bool reset` (which is supplied by the runtime system when the function is invoked) specifies whether the counter value should be reset after evaluating the current value (if applicable).

For instance, here is such a function returning how often it was invoked:

```cpp
// The atomic variable 'counter' ensures the thread safety of the counter.
boost::atomic<std::int64_t> counter(0);

std::int64_t some_performance_data(bool reset)
{
    std::int64_t result = ++counter;
    if (reset)
        counter = 0;
    return result;
}
```

This example function exposes a linearly increasing value as our performance data. The value is incremented on each invocation, e.g. each time a consumer requests the counter data of this Performance Counter.

The next step in exposing this counter to the runtime system is to register the function as a new raw counter type using the HPX API function `hpx::performance_counters::install_counter_type`. A counter type represents certain common characteristics of counters, like their counter type name, and any associated description information. The following snippet shows an example of how to register the function `some_performance_data` which is shown above for a counter type named "/test/data". This registration has to be executed before any consumer instantiates and queries an instance of this counter type:

```cpp
#include <hpx/include/performance_counters.hpp>

void register_counter_type()
{
    // Call the HPX API function to register the counter type.
    hpx::performance_counters::install_counter_type("/test/data",
        &some_performance_data, // function providing counter data
        "returns a linearly increasing counter value", // description text (optional)
        "" // unit of measure (optional)
    );
}
```

Now it is possible to instantiate a new counter instance based on the naming scheme "/test{locality#*/total}/data" where * is a zero based integer index identifying the locality for which the counter instance should
be accessed. The function `hpx::performance_counters::install_counter_type` enables to instantiate exactly one counter instance for each `locality`. Repeated requests to instantiate such a counter will return the same instance, e.g. the instance created for the first request.

If this counter needs to be accessed using the standard `HPX` command line options, the registration has to be performed during application startup, before `hpx_main` is executed. The best way to achieve this is to register an `HPX` startup function using the API function `hpx::register_startup_function` before calling `hpx::init` to initialize the runtime system:

```cpp
int main(int argc, char* argv[]) {
    // By registering the counter type we make it available to any consumer
    // who creates and queries an instance of the type "/test/data".
    // This registration should be performed during startup. The
    // function 'register_counter_type' should be executed as an HPX thread right
    // before hpx_main is executed.
    hpx::register_startup_function(sregister_counter_type);

    // Initialize and run HPX.
    return hpx::init(argc, argv);
}
```

Please see the code in [hpx_link examples/performance_counters/simplest_performance_counter.cpp..simplest_performance_counter.cpp](#) for a full example demonstrating this functionality.

**Implementing a full performance counter**

Sometimes, the simple way of exposing a single value as a Performance Counter is not sufficient. For that reason, `HPX` provides a means of implementing full Performance Counters which support:

- Retrieving the descriptive information about the Performance Counter
- Retrieving the current counter value
- Resetting the Performance Counter (value)
- Starting the Performance Counter
- Stopping the Performance Counter
- Setting the (initial) value of the Performance Counter

Every full Performance Counter will implement a predefined interface:

```cpp
// Copyright (c) 2007-2018 Hartmut Kaiser

#include <hpx/config.hpp>
#include <hpx/lcos/future.hpp>
#include <hpx/runtime/components/client_base.hpp>
#include <hpx/runtime/launch_policy.hpp>
#include <hpx/functional/bind_front.hpp>

// Distributed under the Boost Software License, Version 1.0. (See accompanying
// file LICENSE_1_0.txt or copy at http://www.boost.org/LICENSE_1_0.txt)

#if !defined(HPX_PERFORMANCE_COUNTERS_PERFORMANCE_COUNTER_JAN_18_2013_0939AM)
#define HPX_PERFORMANCE_COUNTERS_PERFORMANCE_COUNTER_JAN_18_2013_0939AM

#include <hpx/config.hpp>
#include <hpx/lcos/future.hpp>
#include <hpx/runtime/components/client_base.hpp>
#include <hpx/runtime/launch_policy.hpp>
#include <hpx/functional/bind_front.hpp>

(continues on next page)
```
#include <hpx/performance_counters/counters_fwd.hpp>
#include <hpx/performance_counters/stubs/performance_counter.hpp>
#include <string>
#include <utility>
#include <vector>

namespace hpx { namespace performance_counters {

struct HPX_EXPORT performance_counter : components::client_base<
  performance_counter, stubs::performance_counter>
{
    typedef components::client_base<
      performance_counter, stubs::performance_counter>
    base_type;

    performance_counter() {}
    performance_counter(std::string const& name);
    performance_counter(std::string const& name,
      hpx::id_type const& locality);
    performance_counter(future<id_type> && id)
      : base_type(std::move(id))
    {}
    performance_counter(hpx::future<
      performance_counter> && c)
      : base_type(std::move(c))
    {}

    future<counter_info> get_info() const;
    counter_info get_info(launch::sync_policy,
      error_code& ec = throws) const;

    future<counter_value> get_counter_value(bool reset = false);
    counter_value get_counter_value(launch::sync_policy,
      bool reset = false, error_code& ec = throws);

    future<counter_value> get_counter_value() const;
    counter_value get_counter_value(launch::sync_policy,
      error_code& ec = throws) const;

    future<counter_values_array> get_counter_values_array(bool reset = false);
    counter_values_array get_counter_values_array(launch::sync_policy,
      bool reset = false, error_code& ec = throws);

    future<counter_values_array> get_counter_values_array() const;
    counter_values_array get_counter_values_array(launch::sync_policy,
      error_code& ec = throws) const;

    future<bool> start();
    bool start(launch::sync_policy, error_code& ec = throws);
future<bool> stop();
bool stop(launch::sync_policy, error_code& ec = throws);

future<void> reset();
void reset(launch::sync_policy, error_code& ec = throws);

future<void> reinit(bool reset = true);
void reinit(
    launch::sync_policy, bool reset = true, error_code& ec = throws);

////////////////////////////////////////////////////////////////////////////////////
future<std::string> get_name() const;
std::string get_name(launch::sync_policy, error_code& ec = throws) const;

private:
    template <typename T>
    static T extract_value(future<counter_value> && value)
    {
        return value.get().get_value<T>();
    }

public:
    template <typename T>
    future<T> get_value(bool reset = false)
    {
        return get_counter_value(reset).then(
            hpx::launch::sync,
            util::bind_front(
                &performance_counter::extract_value<T>));
    }

    template <typename T>
    T get_value(launch::sync_policy, bool reset = false,
        error_code& ec = throws)
    {
        return get_counter_value(launch::sync, reset).get_value<T>(ec);
    }

    template <typename T>
    future<T> get_value() const
    {
        return get_counter_value().then(
            hpx::launch::sync,
            util::bind_front(
                &performance_counter::extract_value<T>));
    }

    template <typename T>
    T get_value(launch::sync_policy, error_code& ec = throws) const
    {
        return get_counter_value(launch::sync).get_value<T>(ec);
    }

    /// Return all counters matching the given name (with optional wildcards).
    HPX_API_EXPORT std::vector<performance_counter> discover_counters(
        std::string const& name, error_code& ec = throws);
    
}
In order to implement a full Performance Counter you have to create an *HPX* component exposing this interface. To simplify this task, *HPX* provides a ready made base class which handles all the boiler plate of creating a component for you. The remainder of this section will explain the process of creating a full Performance Counter based on the Sine example which you can find in the directory examples/performance_counters/sine/.

The base class is defined in the header file [hpx_link hpx/performance_counters/base_performance_counter.hpp](hpx/performance_counters/base_performance_counter.hpp) as:

```cpp
namespace hpx {
  namespace performance_counters {
    template<typename Derived>
    class base_performance_counter;
  }
}
```

```cpp
namespace hpx {
  namespace performance_counters {
    template<typename Derived>
    class base_performance_counter:
      public hpx::performance_counters::server::base_performance_counter,
      public hpx::components::component_base<Derived> {
    private:
      typedef hpx::components::component_base<Derived> base_type;

    public:
      typedef Derived type_holder;
      typedef hpx::performance_counters::server::base_performance_counter
        base_type_holder;

    base_performance_counter()
    {}

    base_performance_counter(hpx::performance_counters::counter_info const&
      info) : base_type_holder(info)
    {}
  }
}
```

(continues on next page)
The single template parameter is expected to receive the type of the derived class implementing the Performance Counter. In the Sine example this looks like:

```cpp
namespace performance_counters {
    namespace sine {
        namespace server {
            //////////////////////////////////////////////////////////////////////
            // [sine_counter_definition
            class sine_counter :
                public hpx::performance_counters::base_performance_counter<sine_counter> {
                
                public:
                    sine_counter() : current_value_(0), evaluated_at_(0) {}
                    sine_counter(hpx::performance_counters::counter_info const & info);

                    /// This function will be called in order to query the current value of
                    /// this performance counter
                    hpx::performance_counters::counter_value get_counter_value(bool reset);

                    /// The functions below will be called to start and stop collecting
                    /// counter values from this counter.
                    bool start();
                    bool stop();

                    /// finalize() will be called just before the instance gets destructed
                    void finalize();

                protected:
                    // Disambiguate finalize() which is implemented in both base classes
                    void finalize() {
                        base_type_holder::finalize();
                        base_type::finalize();
                    }
                }
            }
        }
    }
}
```
i.e. the type `sine_counter` is derived from the base class passing the type as a template argument (please see [hpx_link examples/performance_counters/sine/server/sine.hpp..sine.hpp] for the full source code of the counter definition). For more information about this technique (called Curiously Recurring Template Pattern - CRTP), please see for instance the corresponding Wikipedia article\(^\text{207}\). This base class itself is derived from the `performance_counter` interface described above.

Additionally, a full Performance Counter implementation not only exposes the actual value but also provides information about

- The point in time a particular value was retrieved
- A (sequential) invocation count
- The actual counter value
- An optional scaling coefficient
- Information about the counter status

### Existing HPX performance counters

The HPX runtime system exposes a wide variety of predefined Performance Counters. These counters expose critical information about different modules of the runtime system. They can help determine system bottlenecks and fine-tune system and application performance.

### Table 2.31: AGAS performance counters

<table>
<thead>
<tr>
<th>Counter type</th>
<th>Counter instance formatting</th>
<th>Description</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>/agas/count/</code>&lt;br&gt;<code>&lt;agas_service&gt;</code>&lt;br&gt;<code>where:</code>&lt;br&gt;<code>&lt;agas_service&gt;</code> is one of the following: primary names-space services: route, bind_gid, resolve_gid, unbind_gid, increment_credit, decrement_credit, allocate, begin_migration, end_migration component names-space services: bind_prefix, bind_name, resolve_id, unbind_name, iterate_types, get_component_types, num_localities_type locality namespace services: free, localities, num_localities, num_threads, resolve_locality, resolved_localities symbol namespace services: bind, resolve, unbind, iterate_names, on_symbol_namespace_event</td>
<td><code>&lt;agas_instance&gt;/total</code>&lt;br&gt;<code>where:</code>&lt;br&gt;<code>&lt;agas_instance&gt;</code> is the name of the AGAS service to query. Currently, this value will be <code>locality#0</code> where 0 is the root <code>locality</code> (the id of the locality hosting the AGAS service). The value for <code>*</code> can be any <code>locality id</code> for the following <code>&lt;agas_service&gt;</code>: route, bind_gid, resolve_gid, unbind_gid, increment_credit, decrement_credit, bin, resolve, unbind, and iterate_names only the primary and symbol AGAS service components live on all localities, whereas all other AGAS services are available on <code>locality#0 only</code>.</td>
<td>None</td>
<td>Returns the total number of invocations of the specified AGAS service since its creation.</td>
</tr>
<tr>
<td><code>/agas/</code>&lt;br&gt;<code>&lt;agas_service_category&gt;/count</code>&lt;br&gt;<code>where:</code>&lt;br&gt;<code>&lt;agas_service_category&gt;</code> is one of the following: primary, locality, component or symbol</td>
<td><code>&lt;agas_instance&gt;</code></td>
<td>None</td>
<td>Returns the overall total number of invocations of all AGAS services provided by the given AGAS service category since its creation.</td>
</tr>
</tbody>
</table>

**2.5. Manual**

AGAS service components live on all localities, whereas all other AGAS services are available on
<table>
<thead>
<tr>
<th>Counter type</th>
<th>Description</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>/data/count/ &lt;connection_type&gt; &lt;operation&gt; where: &lt;operation&gt; is one of the following: sent, received &lt;connection_type&gt; is one of the following: tcp, mpi</td>
<td>locality** local** total where: * is the locality id of the locality the overall number of transmitted bytes should be queried for. The locality id is a (zero based) number identifying the locality. Returns the overall number of raw (uncompressed) bytes sent or received (see &lt;operation&gt;, e.g. en or received) for the specified &lt;connection_type&gt;. The performance counters for the connection type mpi are available only if the compile time constant HPX_HAVE_PARCELPORT_MPI was defined while compiling the HPX core library (which is not defined by default, the corresponding cmake configuration constant is HPX WITH PARCELPORT_MPI). Please see CMake variables used to configure HPX for more details.</td>
<td>None</td>
</tr>
<tr>
<td>/data/time/ &lt;connection_type&gt; &lt;operation&gt; where: &lt;operation&gt; is one of the following: sent, received &lt;connection_type&gt; is one of the following: tcp, mpi</td>
<td>locality** local** total where: * is the locality id of the locality the total transmission time should be queried for. The locality id is a (zero based) number identifying the locality. Returns the total time (in nanoseconds) between the start of each asynchronous transmission operation and the end of the corresponding operation for the specified &lt;connection_type&gt; the given locality (see &lt;operation&gt;, e.g. en or received). The performance counters for the connection type mpi are available only if the compile time constant HPX_HAVE_PARCELPORT_MPI was defined while compiling the HPX core library (which is not defined by default, the corresponding cmake configuration constant is HPX WITH PARCELPORT_MPI). Please see CMake variables used to configure HPX for more details.</td>
<td>None</td>
</tr>
<tr>
<td>/serialize/ count/ &lt;connection_type&gt; &lt;operation&gt; where: &lt;operation&gt; is one of the following: sent, received &lt;connection_type&gt; is one of the following: tcp, mpi</td>
<td>locality** local** total where: * is the locality id of the locality the overall number of transmitted bytes should be queried for. The locality id is a (zero based) number identifying the locality. Returns the overall number of bytes transferred (see &lt;operation&gt;, e.g. sent or received possibly compressed) for the specified &lt;connection_type&gt; by the given locality. The performance counters for the connection type mpi are available only if the compile time constant HPX_HAVE_PARCELPORT_MPI was defined while compiling the HPX core library (which is not defined by default, the corresponding cmake configuration constant is HPX WITH PARCELPORT_MPI). Please see CMake variables used to configure HPX for more details.</td>
<td>If the configure-time option -DHPX_WITH_PARCELPORT_ACTION_COUNTERS=On was specified, this counter allows to specify an optional action name as its parameter. In this case the counter will report the number of bytes transmitted for the given action only.</td>
</tr>
</tbody>
</table>
| /serialize/ time/ <connection_type> <operation> where: <operation> is one of the following: sent, received <connection_type> is one of the following: tcp, mpi | locality** local** total where: * is the locality id of the locality Returns the overall time spent performing outgoing data serialization for the specified <connection_type> on the given locality (see <operation>, e.g. sent or received). The performance counters for the connection type mpi are available only if the compile time constant HPX_HAVE_PARCELPORT_MPI was defined while compiling the HPX core library (which is not defined by default, the corresponding cmake configuration constant is HPX WITH PARCELPORT_MPI). Please see CMake variables used to configure HPX for more details. | If the configure-time option -DHPX_WITH_PARCELPORT_ACTION_COUNTERS=On was specified, this counter allows to
Table 2.33: Thread manager performance counters

<table>
<thead>
<tr>
<th>Counter type</th>
<th>Counter instance formatting</th>
<th>Description</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>/threads/count/cumulative</td>
<td>locality#/total or locality#/worker-thread#* or locality#/pool#/worker-thread#*</td>
<td>Returns the overall number of executed (retired) HPX-threads on the given locality since application start. If the instance name is total the counter returns the accumulated number of retired HPX-threads for all worker threads (cores) on that locality. If the instance name is worker-thread#* the counter will return the overall number of retired HPX-threads for all worker threads separately. This counter is available only if the configuration time constant HPX_WITH_THREAD_CUMULATIVE_COUNTS is set to ON (default: ON).</td>
<td>None</td>
</tr>
</tbody>
</table>

Continued on next page

---

208 A message can potentially consist of more than one parcel.
Table 2.33 – continued from previous page

| /threads/time/average | locality#*/total or locality#*/worker-thread#* or locality#*/pool#*/worker-thread#* where: locality#* is defining the locality for which the average time spent executing one HPX-thread should be queried for. The locality id (given by *) is a (zero based) number identifying the locality. pool#* is defining the pool for which the current value of the idle-loop counter should be queried for. worker-thread#* is defining the worker thread for which the average time spent executing one HPX-thread should be queried for. The worker thread number (given by the *) is a (zero based) number identifying the worker thread. The number of available worker threads is usually specified on the command line for the application using the option --hpx:threads. If no pool-name is specified the counter refers to the 'default' pool. | Returns the average time spent executing one HPX-thread on the given locality since application start. If the instance name is total the counter returns the average time spent executing one HPX-thread for all worker threads (cores) on that locality. If the instance name is worker-thread#* the counter will return the average time spent executing one HPX-thread for all worker threads separately. This counter is available only if the configuration time constants HPX_WITH_THREAD_CUMULATIVE_COUNTS (default: ON) and HPX_WITH_THREAD_IDLE_RATES are set to ON (default: OFF). The unit of measure for this counter is nanosecond [ns]. | None |

Continued on next page
<table>
<thead>
<tr>
<th>Threads/Time/Average-Overhead</th>
<th>Returns the average time spent on overhead while executing one HPX-thread on the given locality since application start. If the instance name is total the counter returns the average time spent on overhead while executing one HPX-thread for all worker threads (cores) on that locality. If the instance name is worker-thread#* the counter will return the average time spent on overhead executing one HPX-thread for all worker threads separately. This counter is available only if the configuration time constants HPX_WITH_THREAD_CUMULATIVE_COUNTS (default: ON) and HPX_WITH_THREAD_IDLE_RATES are set to ON (default: OFF). The unit of measure for this counter is nanosecond [ns].</th>
<th>None</th>
</tr>
</thead>
<tbody>
<tr>
<td>/threads/time/average-overhead</td>
<td>locality#<em>/total or locality#</em>/worker-thread#* or locality#<em>/pool#</em>/worker-thread#* where: locality#* is defining the locality for which the average overhead spent executing one HPX-thread should be queried for. The locality id (given by <em>) is a (zero based) number identifying the locality. pool#</em> is defining the pool for which the current value of the idle-loop counter should be queried for. worker-thread#* is defining the worker thread for which the average overhead spent executing one HPX-thread should be queried for. The worker thread number (given by the *) is a (zero based) number identifying the worker thread. The number of available worker threads is usually specified on the command line for the application using the option --hpx:threads. If no pool-name is specified the counter refers to the 'default' pool.</td>
<td>None</td>
</tr>
</tbody>
</table>
Table 2.33 – continued from previous page

| /threads/count/cumulative-phases | Returns the overall number of executed HPX-thread phases (invocations) on the given locality since application start. If the instance name is total the counter returns the accumulated number of executed HPX-thread phases (invocations) for all worker threads (cores) on that locality. If the instance name is worker-thread#* the counter will return the overall number of executed HPX-thread phases for all worker threads separately. This counter is available only if the configuration time constant HPX_WITH_THREAD_CUMULATIVE_COUNTS is set to ON (default: ON). The unit of measure for this counter is nanosecond [ns]. | None |

| locality#*/total or locality#/p | locality#/w | worker-thread#* or locality#/p worker-thread#* where: locality#* is defining the locality for which the overall number of executed HPX-thread phases (invocations) should be queried for. The locality id (given by * is a (zero based) number identifying the locality. pool#* is defining the pool for which the current value of the idle-loop counter should be queried for. worker-thread#* is defining the worker thread for which the overall number of executed HPX-thread phases (invocations) should be queried for. The worker thread number (given by the * is a (zero based) number identifying the worker thread. The number of available worker threads is usually specified on the command line for the application using the option --hpthreads. If no pool-name is specified the counter refers to the ‘default’ pool. |

Continued on next page
Table 2.33 – continued from previous page

| /threads/time/average-phase | locality#*/total or locality#*/worker-thread#* or locality#*/pool#*/worker-thread#* where: locality#* is defining the locality for which the average time spent executing one HPX-thread phase (invocation) should be queried for. The locality id (given by *) is a (zero based) number identifying the locality. pool#* is defining the pool for which the current value of the idle-loop counter should be queried for. worker-thread#* is defining the worker thread for which the average time executing one HPX-thread phase (invocation) should be queried for. The worker thread number (given by the * is a (zero based) number identifying the worker thread. The number of available worker threads is usually specified on the command line for the application using the option --hpx:threads. If no pool-name is specified the counter refers to the 'default' pool. | Returns the average time spent executing one HPX-thread phase (invocation) on the given locality since application start. If the instance name is total the counter returns the average time spent executing one HPX-thread phase (invocation) for all worker threads (cores) on that locality. If the instance name is worker-thread#* the counter will return the average time spent executing one HPX-thread phase for all worker threads separately. This counter is available only if the configuration time constants HPX_WITH_THREAD_CUMULATIVE_COUNTS (default: ON) and HPX_WITH_THREAD_IDLE_RATES are set to ON (default: OFF). The unit of measure for this counter is nanosecond [ns]. | None |
| /threads/time/average-phase-overhead | locality#*/total or locality#*/worker-thread#* or locality#*/pool#*/worker-thread#* where: locality#* is defining the locality for which the average time overhead executing one HPX-thread phase (invocation) should be queried for. The locality id (given by *) is a (zero based) number identifying the locality. pool#* is defining the pool for which the current value of the idle-loop counter should be queried for. worker-thread#* is defining the worker thread for which the average overhead executing one HPX-thread phase (invocation) should be queried for. The worker thread number (given by the *) is a (zero based) number identifying the worker thread. The number of available worker threads is usually specified on the command line for the application using the option --hpx:threads. If no pool-name is specified the counter refers to the ‘default’ pool. | Returns the average time spent on overhead executing one HPX-thread phase (invocation) on the given locality since application start. If the instance name is total the counter returns the average time spent on overhead while executing one HPX-thread phase (invocation) for all worker threads (cores) on that locality. If the instance name is worker-thread#* the counter will return the average time spent on overhead executing one HPX-thread phase for all worker threads separately. This counter is available only if the configuration time constants HPX_WITH_THREAD_CUMULATIVE_COUNTS (default: ON) and HPX_WITH_THREAD_IDLE_RATES are set to ON (default: OFF). The unit of measure for this counter is nanosecond [ns]. | None |

Continued on next page
Table 2.33 – continued from previous page

<table>
<thead>
<tr>
<th>Instance Name</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>/threads/time/overall locality#<em>/total or locality#</em>/worker-thread#<em>/ or locality#</em>/pool#<em>/worker-thread#</em></td>
<td>Returns the overall time spent running the scheduler on the given locality since application start. If the instance name is total the counter returns the overall time spent running the scheduler for all worker threads (cores) on that locality. If the instance name is worker-thread#* the counter will return the overall time spent running the scheduler for all worker threads separately. This counter is available only if the configuration time constant HPX_WITH_THREAD_IDLE_RATES is set to ON (default: OFF). The unit of measure for this counter is nanosecond [ns].</td>
<td>None</td>
</tr>
<tr>
<td>/threads/time/overall locality#*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>/threads/time/overall pool#*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>/threads/time/overall worker-thread#*</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Continued on next page
Table 2.33 – continued from previous page

| /threads/time/cumulative | locality#*/total or locality#*/ worker-thread#* or locality#*/ pool#*/ worker-thread#* where: locality#* is defining the locality for which the overall time spent executing all HPX-threads should be queried for. The locality id (given by *) is a (zero based) number identifying the locality. pool#* is defining the pool for which the current value of the idle-loop counter should be queried for. worker-thread#* is defining the worker thread for which the overall time spent executing all HPX-threads should be queried for. The worker thread number (given by the *) is a (zero based) number identifying the worker thread. The number of available worker threads is usually specified on the command line for the application using the option --hpx:threads. If no pool-name is specified the counter refers to the ‘default’ pool. | Returns the overall time spent executing all HPX-threads on the given locality since application start. If the instance name is total the counter returns the overall time spent executing all HPX-threads for all worker threads (cores) on that locality. If the instance name is worker-thread#* the counter will return the overall time spent executing all HPX-threads for all worker threads separately. This counter is available only if the configuration time constants HPX_THREAD_MAINTAIN_CUMULATIVE_COUNTS (default: ON) and HPX_THREAD_MAINTAIN_IDLE_RATES are set to ON (default: OFF). | None |

Continued on next page
Table 2.33 – continued from previous page

<table>
<thead>
<tr>
<th>Instance</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>locality#/total</td>
<td>Returns the overall overhead time incurred executing all HPX-threads on the given locality since application start. If the instance name is total the counter returns the overall overhead time incurred executing all HPX-threads for all worker threads (cores) on that locality. If the instance name is worker-thread/# the counter will return the overall overhead time incurred executing all HPX-threads for all worker threads separately.</td>
</tr>
<tr>
<td>worker-thread/# or locality#/pool#</td>
<td>This counter is available only if the configuration time constants HPX_THREAD_MAINTAIN_CUMULATIVE_COUNTS (default: ON) and HPX_THREAD_MAINTAIN_IDLE_RATES are set to ON (default: OFF). The unit of measure for this counter is nanosecond [ns].</td>
</tr>
<tr>
<td>pool#</td>
<td>None</td>
</tr>
</tbody>
</table>

Continued on next page
Table 2.33 – continued from previous page

| threads/count/instantaneous/<thread-state> | locality#/total or locality#/worker-thread#/ | Returns the current number of HPX-threads having the given thread state on the given locality. If the instance name is total the counter returns the current number of HPX-threads of the given state for all worker threads (cores) on that locality. If the instance name is worker-thread#* the counter will return the current number of HPX-threads in the given state for all worker threads separately. | None |
| <thread-state> is one of the following: all, active, pending, suspended, terminated, staged | locality#/ or locality#/worker-thread#/ | | |

where:

locality#* is defining the locality for which the current number of threads with the given state should be queried for. The locality id (given by *) is a (zero based) number identifying the locality.

pool#/* is defining the pool for which the current value of the idle-loop counter should be queried for.

worker-thread#* is defining the worker thread for which the current number of threads with the given state should be queried for. The worker thread number (given by the *) is a (zero based) number identifying the worker thread. The number of available worker threads is usually specified on the command line for the application using the option --hpx:threads. If no pool-name is specified the counter refers to the ‘default’ pool.

The staged thread state refers to registered tasks before they are converted to thread objects.

Continued on next page
Table 2.33 – continued from previous page

| threads/wait-time/ | locality#*/total or locality#*/worker-thread### | Returns the average wait time of HPX-threads (if the thread state is pending or of task descriptions (if the thread state is staged) on the given locality since application start. If the instance name is total the counter returns the wait time of HPX-threads of the given state for all worker threads (cores) on that locality. If the instance name is worker-thread### the counter will return the wait time of HPX-threads in the given state for all worker threads separately. These counters are available only if the compile time constant HPX_WITH_THREAD_QUEUE_WAITTIME was defined while compiling the HPX core library (default: OFF). The unit of measure for this counter is nanosecond [ns]. |
| <thread-state> | or locality#*/pool###/worker-thread### | None |
| where: | where: | |
| <thread-state> is one of the following: pending staged | locality#* is defining the locality for which the average wait time of HPX-threads (pending) or thread descriptions (staged) with the given state should be queried for. The locality id (given by * is a (zero based) number identifying the locality. pool#* is defining the pool for which the current value of the idle-loop counter should be queried for. worker-thread#* is defining the worker thread for which the average wait time for the given state should be queried for. The worker thread number (given by the * is a (zero based) number identifying the worker thread. The number of available worker threads is usually specified on the command line for the application using the option --hpx:threads. If no pool-name is specified the counter refers to the ‘default’ pool. The staged thread state refers to the wait time of registered tasks before they are converted into thread objects, while the pending thread state refers to the wait time of threads in any of the scheduling queues. |
| Continued on next page | | |
Table 2.33 – continued from previous page

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/threads/idle-rate</td>
<td>Returns the average idle rate for the given worker thread(s) on the given locality. The idle rate is defined as the ratio of the time spent on scheduling and management tasks and the overall time spent executing work since the application started. This counter is available only if the configuration time constant HPX_WITH_THREAD_IDLE_RATES is set to ON (default: OFF).</td>
</tr>
<tr>
<td>locality#*/total</td>
<td></td>
</tr>
<tr>
<td>locality#*/</td>
<td></td>
</tr>
<tr>
<td>worker-thread#*</td>
<td></td>
</tr>
<tr>
<td>or</td>
<td></td>
</tr>
<tr>
<td>locality#*/</td>
<td></td>
</tr>
<tr>
<td>pool#*/</td>
<td></td>
</tr>
<tr>
<td>worker-thread#*</td>
<td></td>
</tr>
<tr>
<td>where:</td>
<td></td>
</tr>
<tr>
<td>locality#*</td>
<td>is defining the locality for which the average idle rate of all (or one) worker threads should be queried for. The locality id (given by *) is a (zero based) number identifying the locality.</td>
</tr>
<tr>
<td>pool#*</td>
<td>is defining the pool for which the current value of the idle-loop counter should be queried for.</td>
</tr>
<tr>
<td>worker-thread#*</td>
<td>is defining the worker thread for which the averaged idle rate should be queried for. The worker thread number (given by the *) is a (zero based) number identifying the worker thread. The number of available worker threads is usually specified on the command line for the application using the option --hpx:threads. If no pool-name is specified the counter refers to the ‘default’ pool.</td>
</tr>
</tbody>
</table>
Table 2.33 – continued from previous page

| /threads/creation-idle-rate | locality#/total or locality#*/worker-thread#* or locality#/pool#/worker-thread#* where: locality#/ is defining the locality for which the average creation idle rate of all (or one) worker threads should be queried for. The locality id (given by * is a (zero based) number identifying the locality. pool#/ is defining the pool for which the current value of the idle-loop counter should be queried for. worker-thread#/ is defining the worker thread for which the averaged idle rate should be queried for. The worker thread number (given by the * is a (zero based) number identifying the worker thread. The number of available worker threads is usually specified on the command line for the application using the option --hpx:threads. If no pool-name is specified the counter refers to the 'default' pool. | None |

Returns the average idle rate for the given worker thread(s) on the given locality which is caused by creating new threads. The creation idle rate is defined as the ratio of the time spent on creating new threads and the overall time spent executing work since the application started. This counter is available only if the configuration time constants HPX_WITH_THREAD_IDLE_RATES (default: OFF) and HPX_WITH_THREAD_CREATION_AND_CLEANUP_RATES are set to ON.

Continued on next page
### Table 2.33 – continued from previous page

<table>
<thead>
<tr>
<th>code</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>/threads/</td>
<td>Returns the average idle rate for the given worker thread(s) on the given locality which is caused by cleaning up terminated threads. The cleanup idle rate is defined as the ratio of the time spent on cleaning up terminated thread objects and the overall time spent executing work since the application started. This counter is available only if the configuration time constants <code>HPX_WITH_THREAD_IDLE_RATES</code> (default: OFF) and <code>HPX_WITH_THREAD_CREATION_AND_CLEANUP_RATES</code> are set to ON.</td>
<td>None</td>
</tr>
<tr>
<td>cleanup-idle-rate</td>
<td></td>
<td></td>
</tr>
<tr>
<td>locality#/total</td>
<td></td>
<td></td>
</tr>
<tr>
<td>or</td>
<td></td>
<td></td>
</tr>
<tr>
<td>locality#/</td>
<td></td>
<td></td>
</tr>
<tr>
<td>worker-thread#*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>or</td>
<td></td>
<td></td>
</tr>
<tr>
<td>locality#/*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pool#*/</td>
<td></td>
<td></td>
</tr>
<tr>
<td>worker-thread#*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>where:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>locality#/*</td>
<td>is defining the locality for which the average cleanup idle rate of all (or one) worker threads should be queried for. The locality id (given by *) is a (zero based) number identifying the locality.</td>
<td></td>
</tr>
<tr>
<td>pool#/*</td>
<td>is defining the pool for which the current value of the idle-loop counter should be queried for.</td>
<td></td>
</tr>
<tr>
<td>worker-thread#*</td>
<td>is defining the worker thread for which the averaged cleanup idle rate should be queried for. The worker thread number (given by the *) is a (zero based) number identifying the worker thread. The number of available worker threads is usually specified on the command line for the application using the option <code>--hpx:threads</code>. If no pool-name is specified the counter refers to the ‘default’ pool.</td>
<td></td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>RETURNS</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>/threadqueue/length</code></td>
<td>Returns the overall length of all queues for the given worker thread(s) on</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>the given locality.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>/threads/count/stack-unbinds</code></td>
<td>Returns the total number of HPX-thread unbind (madvise) operations performed</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 2.33 – continued from previous page

<table>
<thead>
<tr>
<th>Path</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>/threads/count/stack-recycles</code></td>
<td>Returns the total number of HPX-thread recycling operations performed.</td>
<td>None</td>
</tr>
<tr>
<td><code>/threads/count/stolen-from-pending</code></td>
<td>Returns the total number of HPX-threads ‘stolen’ from the pending thread queue by a neighboring thread worker thread (these threads are executed by a different worker thread than they were initially scheduled on). This counter is available only if the configuration time constant <code>HPX_WITH_THREAD_STEALING_COUNTS</code> is set to ON (default: ON).</td>
<td>None</td>
</tr>
</tbody>
</table>

Continued on next page
Table 2.33 – continued from previous page

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/threads/count/pending-misses</td>
<td>Returns the total number of times that the referenced worker-thread on the referenced locality failed to find pending HPX-threads in its associated queue. This counter is available only if the configuration time constant HPX_WITH_THREAD_STEALING_COUNTS is set to ON (default: ON).</td>
</tr>
<tr>
<td>locality#/total or locality#<em>/worker-thread#</em> or locality#/pool#/worker-thread#* where: locality#* is defining the locality for which the number of pending queue misses of all (or one) worker threads should be queried for. The locality id (given by <em>) is a (zero based) number identifying the locality pool#</em> is defining the pool for which the current value of the idle-loop counter should be queried for. worker-thread#* is defining the worker thread for which the number of pending queue misses should be queried for. The worker thread number (given by the *) is a (zero based) number identifying the worker thread. The number of available worker threads is usually specified on the command line for the application using the option --hpx:threads. If no pool-name is specified the counter refers to the ‘default’ pool.</td>
<td>None</td>
</tr>
</tbody>
</table>

Continued on next page
---

### Table 2.33 – continued from previous page

<table>
<thead>
<tr>
<th>/threads/count/pending-accesses</th>
<th>locality#<em>/total or locality#</em>/worker-thread#* or locality#<em>/pool#</em>/worker-thread#* where:</th>
<th>Returns the total number of times that the referenced worker-thread on the referenced locality looked for pending HPX-threads in its associated queue. This counter is available only if the configuration time constant HPX_WITH_THREAD_STEALING_COUNTS is set to ON (default: ON).</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>locality#* is defining the locality for which the number of pending queue accesses of all (or one) worker threads should be queried for. The locality id (given by *) is a (zero based) number identifying the locality.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>pool#* is defining the pool for which the current value of the idle-loop counter should be queried for.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>worker-thread#* is defining the worker thread for which the number of pending queue accesses should be queried for. The worker thread number (given by the *) is a (zero based) number identifying the worker thread.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>The number of available worker threads is usually specified on the command line for the application using the option --hpx:threads. If no pool-name is specified the counter refers to the ‘default’ pool.</td>
<td></td>
</tr>
</tbody>
</table>

Continued on next page
Table 2.33 – continued from previous page

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>/threads/count/stolen-from-staged</td>
<td>Returns the total number of HPX-threads ‘stolen’ from the staged thread queue by a neighboring worker thread (these threads are executed by a different worker thread than they were initially scheduled on). This counter is available only if the configuration time constant <code>HPX_WITH_THREAD_STEALING_COUNTS</code> is set to <code>ON</code> (default: <code>ON</code>).</td>
<td>None</td>
</tr>
</tbody>
</table>

Continued on next page

/threads/count/stolen-from-staged

locality#/total
or
locality#/worker-thread#*
or
locality#/pool#*/worker-thread#

where:

- `locality#` is defining the locality for which the number of HPX-threads stolen from the staged queue of all (or one) worker threads should be queried for. The `locality` id (given by `*`) is a (zero based) number identifying the `locality`.
- `pool#` is defining the pool for which the current value of the idle-loop counter should be queried for.
- `worker-thread#` is defining the worker thread for which the number of HPX-threads stolen from the staged queue should be queried for. The worker thread number (given by the `*`) is a (zero based) number identifying the worker thread. The number of available worker threads is usually specified on the command line for the application using the option `--hpx:threads`. If no pool-name is specified the counter refers to the ‘default’ pool.
Table 2.33 – continued from previous page

| Threads/Count | Locality#*/total or locality#*/worker-thread#* or locality#*/pool#*/worker-thread#* where: locality#* is defining the locality for which the number of HPX-threads stolen to the pending queue of all (or one) worker threads should be queried for. The locality id (given by *) is a (zero based) number identifying the locality. pool#* is defining the pool for which the current value of the idle-loop counter should be queried for. worker-thread#* is defining the worker thread for which the number of HPX-threads stolen to the pending queue should be queried for. The worker thread number (given by the *) is a (zero based) number identifying the worker thread. The number of available worker threads is usually specified on the command line for the application using the option --hpx:threads. If no pool-name is specified the counter refers to the 'default' pool. | Returns the total number of HPX-threads ‘stolen’ to the pending thread queue of the worker thread (these threads are executed by a different worker thread than they were initially scheduled on). This counter is available only if the configuration time constant HPX_WITH_THREAD_STEALING_COUNTS is set to ON (default: ON). | None |

Continued on next page
Table 2.33 – continued from previous page

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/threads/count/stolen-to-staged</td>
<td>Returns the total number of HPX-threads ‘stolen’ to the staged thread queue of a neighboring worker thread (these threads are executed by a different worker thread than they were initially scheduled on). This counter is available only if the configuration time constant HPX_WITH_THREAD_STEALING_COUNTS is set to ON (default: ON).</td>
</tr>
</tbody>
</table>

None

Continued on next page
Table 2.33 – continued from previous page

<table>
<thead>
<tr>
<th>Path</th>
<th>Description</th>
<th>None</th>
</tr>
</thead>
<tbody>
<tr>
<td>/threads/count/objects</td>
<td><em><em>locality</em>/total</em>* or *<em>locality</em>/*<em>allocator</em>/ where:*<em>locality</em>/ is defining the <strong>locality</strong> for which the current (cumulative) number of all created <strong>HPX</strong>-thread objects should be queried for. The <strong>locality</strong> id (given by *) is a (zero based) number identifying the <strong>locality</strong>. *<em>allocator</em>/ is defining the number of the allocator instance using which the threads have been created. <strong>HPX</strong> uses a varying number of allocators to create (and recycle) <strong>HPX</strong>-thread objects, most likely these counters are of use for debugging purposes only. The allocator id (given by *) is a (zero based) number identifying the allocator to query.</td>
<td>Returns the total number of <strong>HPX</strong>-thread objects created. Note that thread objects are reused to improve system performance, thus this number does not reflect the number of actually executed (retired) <strong>HPX</strong>-threads.</td>
</tr>
<tr>
<td>/scheduler/utilization/instantaneous</td>
<td><em><em>locality</em>/total</em>* where:*<em>locality</em>/ is defining the <strong>locality</strong> for which the current (instantaneous) scheduler utilization queried for. The <strong>locality</strong> id (given by *) is a (zero based) number identifying the <strong>locality</strong>.</td>
<td>Returns the total (instantaneous) scheduler utilization. This is the current percentage of scheduler threads executing <strong>HPX</strong> threads.</td>
</tr>
</tbody>
</table>
Table 2.33 – continued from previous page

| Threads/Idle-loop-count/Instantaneous | locality#* / worker-thread#* or locality#* / pool#* / worker-thread#* where: locality#* is defining the locality for which the current accumulated value of all idle-loop counters of all worker threads should be queried. The locality id (given by * is a (zero based) number identifying the locality. pool#* is defining the pool for which the current value of the idle-loop counter should be queried for. worker-thread#* is defining the worker thread for which the current value of the idle-loop counter should be queried for. The worker thread number (given by the * is a (zero based) worker thread number (given by the * is a (zero based) number identifying the worker thread. The number of available worker threads is usually specified on the command line for the application using the option --hpx:threads. If no pool-name is specified the counter refers to the ‘default’ pool. | Returns the current (instantaneous) idle-loop count for the given HPX-worker thread or the accumulated value for all worker threads. | None |

Continued on next page
Table 2.33 – continued from previous page

<table>
<thead>
<tr>
<th>/threads/</th>
<th>locality#/*</th>
<th>Returns the current (instantaneous) busy-loop count for the given HPX-worker thread or the accumulated value for all worker threads.</th>
<th>worker-thread#/*</th>
<th>pool#/*</th>
<th>None</th>
</tr>
</thead>
<tbody>
<tr>
<td>busy-loop-count/</td>
<td>locality#/*</td>
<td>or</td>
<td>worker-thread#/*</td>
<td>locality#/*</td>
<td></td>
</tr>
<tr>
<td>instantaneous</td>
<td>pool#/*</td>
<td>worker-thread#/*</td>
<td>where:</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>locality#/* is defining the locality for which the current current accumulated value of all busy-loop counters of all worker threads should be queried. The locality id (given by * is a (zero based) number identifying the locality.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(10pt)</td>
<td>pool#/*</td>
<td>worker-thread#/*</td>
<td>where:</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>locality#/* is defining the pool for which the current value of the idle-loop counter should be queried for.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>worker-thread#/* is defining the worker thread for which the current value of the busy-loop counter should be queried for. The worker thread number (given by the * is a (zero based) worker thread number (given by the * is a (zero based) number identifying the worker thread. The number of available worker threads is usually specified on the command line for the application using the option --hpx:threads. If no pool-name is specified the counter refers to the ‘default’ pool.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Continued on next page
Table 2.33 – continued from previous page

| /threads/time/ | locality#/total | Returns the overall time spent performing background work on the given locality since application start. If the instance name is *total*, the counter returns the overall time spent performing background work for all worker threads (cores) on that locality. If the instance name is *worker-thread#* the counter will return the overall time spent performing background work for all worker threads separately. This counter is available only if the configuration time constants HPX_WITH_BACKGROUND_THREAD_COUNTERS (default: OFF) and HPX_WITH_THREAD_IDLE_RATES are set to ON (default: OFF). The unit of measure for this counter is nanosecond [ns]. |
| background-work-duration | locality#/ | None |
| | worker-thread#* | | where:
| | locality#/ is defining the locality for which the overall time spent performing background work should be queried for. The locality id (given by *) is a (zero based) number identifying the locality. |
| | worker-thread#* is defining the worker thread for which the overall time spent performing background work should be queried for. The worker thread number (given by the *) is a (zero based) number identifying the worker thread. The number of available worker threads is usually specified on the command line for the application using the option --hpx:threads. |

Continued on next page
Table 2.33 – continued from previous page

<table>
<thead>
<tr>
<th>locality#/total or</th>
<th>locality#/worker-thread#/*</th>
<th>Returns the background overhead on the given locality since application start. If the instance name is total the counter returns the background overhead for all worker threads (cores) on that locality. If the instance name is worker-thread#/* the counter will return background overhead for all worker threads separately. This counter is available only if the configuration time constants HPX_WITH_BACKGROUND_THREAD_COUNTERS (default: OFF) and HPX_WITH_THREAD_IDLE_RATES are set to ON (default: OFF). The unit of measure displayed for this counter is 0.1%.</th>
<th>None</th>
</tr>
</thead>
<tbody>
<tr>
<td>/threads/</td>
<td>background-overhead</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>locality#/total</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>or</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>locality#/worker-thread#/*</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>where:</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>locality#/* is defining the locality for which the background overhead should be queried for. The locality id (given by #) is a (zero based) number identifying the locality.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>worker-thread#/* is defining the worker thread for which the background overhead should be queried for. The worker thread number (given by the *) is a (zero based) number identifying the worker thread. The number of available worker threads is usually specified on the command line for the application using the option --hpx:threads.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Continued on next page
Table 2.33 – continued from previous page

| /threads/time/               | locality#/*/total or locality#/*/worker-thread#* where: locality#/* is defining the locality for which the overall time spent performing background work related to sending parcels should be queried for. The locality id (given by *) is a (zero based) number identifying the locality. worker-thread#* is defining the worker thread for which the overall time spent performing background work related to sending parcels should be queried for. The worker thread number (given by the *) is a (zero based) number identifying the worker thread. The number of available worker threads is usually specified on the command line for the application using the option --hpx:threads. | Returns the overall time spent performing background work related to sending parcels on the given locality since application start. If the instance name is total the counter returns the overall time spent performing background work for all worker threads (cores) on that locality. If the instance name is worker-thread#* the counter will return the overall time spent performing background work for all worker threads separately. This counter is available only if the configuration time constants HPX_WITH_BACKGROUND_THREAD_COUNTERS (default: OFF) and HPX_WITH_THREAD_IDLE_RATES are set to ON (default: OFF). The unit of measure for this counter is nanosecond [ns]. This counter will currently return meaningful values for the MPI parcelport only. | None |

Continued on next page
Table 2.33 – continued from previous page

<table>
<thead>
<tr>
<th>Instance</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/threads/background-send-overhead</td>
<td>Returns the background overhead related to sending parcels on the given locality since application start. If the instance name is total, the counter returns the background overhead for all worker threads (cores) on that locality. If the instance name is worker-thread#, the counter will return background overhead for all worker threads separately. This counter is available only if the configuration time constants HPX_WITH_BACKGROUND_THREAD_COUNTERS (default: OFF) and HPX_WITH_THREAD_IDLE_RATES are set to ON (default: OFF). The unit of measure displayed for this counter is 0.1%. This counter will currently return meaningful values for the MPI parcel-port only.</td>
</tr>
<tr>
<td>locality#/total</td>
<td>where: locality#* is defining the locality for which the background overhead related to sending parcels should be queried for. The locality id (given by <em>) is a (zero based) number identifying the locality. worker-thread#</em> is defining the worker thread for which the background overhead related to sending parcels should be queried for. The worker thread number (given by the *) is a (zero based) number identifying the worker thread. The number of available worker threads is usually specified on the command line for the application using the option --hpx:threads.</td>
</tr>
<tr>
<td>None</td>
<td></td>
</tr>
</tbody>
</table>

Continued on next page
Table 2.33 – continued from previous page

| /threads/time/background-receive-duration | Returns the overall time spent performing background work related to receiving parcels on the given locality since application start. If the instance name is total the counter returns the overall time spent performing background work for all worker threads (cores) on that locality. If the instance name is worker-thread#* the counter will return the overall time spent performing background work for all worker threads separately. This counter is available only if the configuration time constants HPX_WITH_BACKGROUND_THREAD_COUNTERS (default: OFF) and HPX_WITH_THREAD_IDLE_RATES are set to ON (default: OFF). The unit of measure for this counter is nanosecond [ns]. This counter will currently return meaningful values for the MPI parcelport only. | None |
| locality#*/total | | |
| locality#*/worker-thread#* | | |
| where: | | |
| locality#* is defining the locality for which the overall time spent performing background work related to receiving parcels should be queried for. The locality id (given by *) is a (zero based) number identifying the locality. | | |
| worker-thread#* is defining the worker thread for which the overall time spent performing background work related to receiving parcels should be queried for. The worker thread number (given by the *) is a (zero based) number identifying the worker thread. The number of available worker threads is usually specified on the command line for the application using the option --hpx:threads. | | |

Continued on next page
### Table 2.33 – continued from previous page

<table>
<thead>
<tr>
<th>Definition</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>locality#* / total overhead</td>
<td>Returns the background overhead related to receiving parcels on the given locality since application start. If the instance name is <code>total</code> the counter returns the background overhead for all worker threads (cores) on that locality. If the instance name is <code>worker-thread#*</code> the counter will return background overhead for all worker threads separately. This counter is available only if the configuration time constants HPX_WITH_BACKGROUND_THREAD_COUNTERS (default: OFF) and HPX_WITH_THREAD_IDLE_RATES are set to ON (default: OFF). The unit of measure displayed for this counter is 0.1%. This counter will currently return meaningful values for the MPI parcel-port only.</td>
<td>None</td>
</tr>
<tr>
<td>locality#* / worker-thread#*</td>
<td>where: <code>locality#*</code> is defining the locality for which the background overhead related to receiving should be queried for. The locality id (given by the *) is a (zero based) number identifying the locality. <code>worker-thread#*</code> is defining the worker thread for which the background overhead related to receiving parcels should be queried for. The worker thread number (given by the *) is a (zero based) number identifying the worker thread. The number of available worker threads is usually specified on the command line for the application using the option <code>--hpx:threads</code>.</td>
<td>None</td>
</tr>
<tr>
<td>/threads/ background-receive-</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 2.34: General performance counters exposing characteristics of localities

<table>
<thead>
<tr>
<th>Counter type</th>
<th>Counter instance formatting</th>
<th>Description</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>/runtime/count/component</td>
<td>locality#/total where: * is the locality id of the locality</td>
<td>Returns the overall number of currently active components of the specified type on the given locality.</td>
<td>The type of the component. This is the string which has been used while registering the component with HPX, e.g. which has been passed as the second parameter to the macro <code>HPX_REGISTER_COMPONENT</code>.</td>
</tr>
<tr>
<td>/runtime/count/action-invocation</td>
<td>locality#/total where: * is the locality id of the locality</td>
<td>Returns the overall (local) invocation count of the specified action type on the given locality.</td>
<td>The action type. This is the string which has been used while registering the action with HPX, e.g. which has been passed as the second parameter to the macro <code>HPX_REGISTER_ACTION</code> or <code>HPX_REGISTER_ACTION_ID</code>.</td>
</tr>
<tr>
<td>/runtime/count/remote-action-invocation</td>
<td>locality#/total where: * is the locality id of the locality</td>
<td>Returns the overall (remote) invocation count of the specified action type on the given locality.</td>
<td>The action type. This is the string which has been used while registering the action with HPX, e.g. which has been passed as the second parameter to the macro <code>HPX_REGISTER_ACTION</code> or <code>HPX_REGISTER_ACTION_ID</code>.</td>
</tr>
<tr>
<td>/runtime/uptime</td>
<td>locality#/total where: * is the locality id of the locality</td>
<td>Returns the overall time since application start on the given locality in nanoseconds.</td>
<td>None</td>
</tr>
<tr>
<td>/runtime/memory/virtual</td>
<td>locality#/total where: * is the locality id of the locality</td>
<td>Returns the amount of virtual memory currently allocated by the referenced locality (in bytes).</td>
<td>None</td>
</tr>
<tr>
<td>/runtime/memory/resident</td>
<td>locality#/total where: * is the locality id of the locality</td>
<td>Returns the amount of resident memory currently allocated by the referenced locality (in bytes).</td>
<td>None</td>
</tr>
</tbody>
</table>

2.5. Manual

/(/runtime/memory/total

Returns the total available memory for use by the referenced locality (in bytes). None
Table 2.35: Performance counters exposing PAPI hardware counters

<table>
<thead>
<tr>
<th>Counter type</th>
<th>Counter instance formatting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/papi/&lt;papi_event&gt;</td>
<td>locality#/total or locality#/worker-thread#*</td>
<td>This counter returns the current count of occurrences of the specified PAPI event. This counter is available only if the configuration time constant <code>HPX_WITH_PAPI</code> is set to ON (default: OFF).</td>
</tr>
<tr>
<td>where:</td>
<td>where:</td>
<td>None</td>
</tr>
<tr>
<td>&lt;papi_event&gt; is the name of the PAPI event to expose as a performance counter (such as <code>PAPI_SR_INS</code>). Note that the list of available PAPI events changes depending on the used architecture. For a full list of available PAPI events and their (short) description use the <code>--hpx:list-counters</code> and <code>--hpx:papi-event-info=all</code> command line options.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>locality#* is defining the locality for which the current current accumulated value of all busy-loop counters of all worker threads should be queried. The <code>locality</code> id (given by <em>) is a (zero based) number identifying the locality. worker-thread#</em> is defining the worker thread for which the current value of the busy-loop counter should be queried for. The worker thread number (given by the *) is a (zero based) worker thread number (given by the *) is a (zero based) number identifying the worker thread. The number of available worker threads is usually specified on the command line for the application using the option <code>--hpx:threads</code>.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Chapter 2. What’s so special about HPX?
Table 2.36: Performance counters for general statistics

<table>
<thead>
<tr>
<th>Counter type</th>
<th>Description</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>/statistics/counter name. The referenced performance counter is queried at fixed time intervals as specified by the first parameter.</td>
<td>Returns the current average (mean) value calculated based on the values queried from the underlying counter (the one specified as the instance name).</td>
<td>Any parameter will be interpreted as a list of up to two comma separated (integer) values, where the first is the time interval (in milliseconds) at which the underlying counter should be queried. If no value is specified, the counter will assume 1000 [ms] as the default. The second value can be either 0 or 1 and specifies whether the underlying counter should be reset during evaluation 1 or not 0. The default value is 0.</td>
</tr>
<tr>
<td>/statistics/counter name. The referenced performance counter is queried at fixed time intervals as specified by the first parameter.</td>
<td>Returns the current rolling average (mean) value calculated based on the values queried from the underlying counter (the one specified as the instance name).</td>
<td>Any parameter will be interpreted as a list of up to three comma separated (integer) values, where the first is the time interval (in milliseconds) at which the underlying counter should be queried. If no value is specified, the counter will assume 1000 [ms] as the default. The second value will be interpreted as the size of the rolling window (the number of latest values to use to calculate the rolling average). The default value for this is 10. The third value can be either 0 or 1 and specifies whether the underlying counter should be reset during evaluation 1 or not 0. The default value is 0.</td>
</tr>
<tr>
<td>Any full performance counter name. The referenced performance counter is queried at fixed time intervals as specified by the first parameter.</td>
<td>Returns the current standard deviation (stddev) value calculated based on the values queried from the underlying counter (the one specified as the instance name).</td>
<td>Any parameter will be interpreted as a list of up to two comma separated (integer) values, where the first is the time interval (in milliseconds) at which the underlying counter should be queried. If no value is specified, the counter will assume 1000 [ms] as the default. The second value can be either 0 or 1 and specifies whether the underlying counter should be reset during evaluation 1 or not 0. The default value is 0.</td>
</tr>
<tr>
<td>Any full performance counter name. The referenced performance counter is queried at fixed time intervals as specified by the first parameter.</td>
<td>Returns the current rolling variance (stddev) value calculated based on the values queried from the underlying counter (the one specified as the instance name).</td>
<td>Any parameter will be interpreted as a list of up to three comma separated (integer) values, where the first is the time interval (in milliseconds) at which the underlying counter should be queried. If no value is specified, the counter will assume 1000 [ms] as the default. The second value will be interpreted as the size of the rolling window (the number of latest values to use to calculate the rolling average). The default value for this is 10. The third value can be either 0 or 1 and specifies whether the underlying counter should be reset during evaluation 1 or not 0. The default value is 0.</td>
</tr>
<tr>
<td>Any full performance counter name. The referenced performance counter is queried at fixed time intervals as specified by the first parameter.</td>
<td>Returns the current (statistically estimated) median value calculated based on the values queried from the underlying counter (the one specified as the instance name).</td>
<td>Any parameter will be interpreted as a list of up to two comma separated (integer) values, where the first is the time interval (in milliseconds) at which the underlying counter should be queried. If no value is specified, the counter will assume 1000 [ms] as the default. The second value can be either 0 or 1 and specifies whether the underlying counter should be reset during evaluation 1 or not 0. The default value is 0.</td>
</tr>
<tr>
<td>Counter type</td>
<td>Counter instance formatting</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>/arithmetics/add</td>
<td>None</td>
<td>Returns the sum calculated based on the values queried from the underlying counters (the ones specified as the parameters).</td>
</tr>
<tr>
<td>/arithmetics/subtract</td>
<td>None</td>
<td>Returns the difference calculated based on the values queried from the underlying counters (the ones specified as the parameters).</td>
</tr>
<tr>
<td>/arithmetics/multiply</td>
<td>None</td>
<td>Returns the product calculated based on the values queried from the underlying counters (the ones specified as the parameters).</td>
</tr>
<tr>
<td>/arithmetics/divide</td>
<td>None</td>
<td>Returns the result of division of the values queried from the underlying counters (the ones specified as the parameters).</td>
</tr>
<tr>
<td>/arithmetics/mean</td>
<td>None</td>
<td>Returns the average value of all values queried from the underlying counters (the ones specified as the parameters).</td>
</tr>
<tr>
<td>/arithmetics/variance</td>
<td>None</td>
<td>Returns the standard deviation of all values queried from the underlying counters (the ones specified as the parameters).</td>
</tr>
<tr>
<td>/arithmetics/median</td>
<td>None</td>
<td>Returns the median value of all values queried from the underlying counters (the ones specified as the parameters).</td>
</tr>
<tr>
<td>/arithmetics/min</td>
<td>None</td>
<td>Returns the minimum value of all values queried from the underlying counters (the ones specified as the parameters).</td>
</tr>
<tr>
<td>/arithmetics/max</td>
<td>None</td>
<td>Returns the maximum value of all values queried from the underlying counters (the ones specified as the parameters).</td>
</tr>
<tr>
<td>/arithmetics/count</td>
<td>None</td>
<td>Returns the count value of all values queried from the underlying counters (the ones specified as the parameters).</td>
</tr>
</tbody>
</table>

**Note:** The /arithmetics counters can consume an arbitrary number of other counters. For this reason those have to be specified as parameters (a comma separated list of counters appended after a ‘@’. For instance:

```
./bin/hello_world_distributed -t2 \
```

(continues on next page)
--hpx:print-counter=/threads{locality#0/worker-thread#*}/count/cumulative \
--hpx:print-counter=/arithmetics/add@/threads{locality#0/worker-thread#*}/count/
→cumulative
hello world from OS-thread 0 on locality 0
hello world from OS-thread 1 on locality 0
/threads{locality#0/worker-thread#0}/count/cumulative,1,0.515640,[s],25
/threads{locality#0/worker-thread#1}/count/cumulative,1,0.515520,[s],36
/arithmetics/add@/threads{locality#0/worker-thread#*}/count/cumulative,1,0.516445,[s],
→64

Since all wildcards in the parameters are expanded, this example is fully equivalent to specifying both counters separately to /arithmetics/add:

./bin/hello_world_distributed -t2 \  
   --hpx:print-counter=/threads{locality#0/worker-thread#*}/count/cumulative \  
   --hpx:print-counter=/arithmetics/add@\  
      /threads{locality#0/worker-thread#0}/count/cumulative,\  
      /threads{locality#0/worker-thread#1}/count/cumulative
Table 2.38: Performance counters tracking parcel coalescing

<table>
<thead>
<tr>
<th>Counter type</th>
<th>Counter instance formatting</th>
<th>Description</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>/coalesce count/parcels/<em>locality</em>/</td>
<td>Returns the number of parcels handled by the message handler associated with the action which is given by the counter parameter.</td>
<td>The action type. This is the string which has been used while registering the action with HPX, e.g. which has been passed as the second parameter to the macro HPX_REGISTER_ACTION or HPX_REGISTER_ACTION_ID.</td>
<td></td>
</tr>
<tr>
<td>/coalesce count/messages/<em>locality</em>/</td>
<td>Returns the number of messages generated by the message handler associated with the action which is given by the counter parameter.</td>
<td>The action type. This is the string which has been used while registering the action with HPX, e.g. which has been passed as the second parameter to the macro HPX_REGISTER_ACTION or HPX_REGISTER_ACTION_ID.</td>
<td></td>
</tr>
<tr>
<td>/coalesce count/average-parcels-per-message/<em>locality</em>/</td>
<td>Returns the average number of parcels sent in a message generated by the message handler associated with the action which is given by the counter parameter.</td>
<td>The action type. This is the string which has been used while registering the action with HPX, e.g. which has been passed as the second parameter to the macro HPX_REGISTER_ACTION or HPX_REGISTER_ACTION_ID.</td>
<td></td>
</tr>
<tr>
<td>/coalesce count/average-parcel-arrival time/<em>locality</em>/</td>
<td>Returns the average time between arriving parcels for the action which is given by the counter parameter.</td>
<td>The action type. This is the string which has been used while registering the action with HPX, e.g. which has been passed as the second parameter to the macro HPX_REGISTER_ACTION or HPX_REGISTER_ACTION_ID.</td>
<td></td>
</tr>
</tbody>
</table>

For each bucket the counter shows a value of the collected histogram, and the number of buckets for the histogram to generate. The first unit of measure displayed for this counter parameter.

The action type. This is the string which has been used while registering the action with HPX, e.g. which has been passed as the second parameter to the macro HPX_REGISTER_ACTION or HPX_REGISTER_ACTION_ID.
Note: The performance counters related to parcel coalescing are available only if the configuration time constant HPX\_WITH\_PARCEL\_COALESCING is set to ON (default: ON). However, even in this case it will be available only for those actions, which are enabled for parcel coalescing (see the macros HPX\_ACTION\_USES\_MESSAGE\_COALESCING and HPX\_ACTION\_USES\_MESSAGE\_COALESCING\_NOTHROW).

APEX integration

HPX provides integration with APEX\(^{209}\), which is a framework for application profiling using task timers and various performance counters. It can be added as a git submodule by turning on the option HPX\_WITH\_APEX:BOOL during CMake\(^{210}\) configuration. TAU\(^{211}\) is an optional dependency when using APEX\(^{212}\).

To build HPX with APEX\(^{213}\) add HPX\_WITH\_APEX=ON, and, optionally, TAU\_ROOT=PATH\_TO\_TAU to your CMake\(^{214}\) configuration. In addition, you can override the tag used for APEX\(^{215}\) with the HPX\_WITH\_APEX\_TAG option. Please see the APEX HPX documentation\(^{216}\) for detailed instructions on using APEX\(^{217}\) with HPX.

2.5.11 HPX runtime and resources

HPX thread scheduling policies

The HPX runtime has five thread scheduling policies: local-priority, static-priority, local, static and abp-priority. These policies can be specified from the command line using the command line option --hpx:queuing. In order to use a particular scheduling policy, the runtime system must be built with the appropriate scheduler flag turned on (e.g. cmake -DHPX\_THREAD\_SCHEDULERS=local, see CMake variables used to configure HPX for more information).

Priority local scheduling policy (default policy)

- default or invoke using: --hpx:queuing=local-priority-fifo

The priority local scheduling policy maintains one queue per operating system (OS) thread. The OS thread pulls its work from this queue. By default the number of high priority queues is equal to the number of OS threads; the number of high priority queues can be specified on the command line using --hpx:high-priority-threads. High priority threads are executed by any of the OS threads before any other work is executed. When a queue is empty work will be taken from high priority queues first. There is one low priority queue from which threads will be scheduled only when there is no other work.

For this scheduling policy there is an option to turn on NUMA sensitivity using the command line option --hpx:numa-sensitive. When NUMA sensitivity is turned on work stealing is done from queues associated with the same NUMA domain first, only after that work is stolen from other NUMA domains.

This scheduler is enabled at build time by default and will be available always.

This scheduler can be used with two underlying queuing policies (FIFO: first-in-first-out, and LIFO: last-in-first-out). The default is FIFO. In order to use the LIFO policy use the command line option --hpx:queuing=local-priority-lifo.

\(^{209}\) https://khuck.github.io/xpress-apex/

\(^{210}\) https://www.cmake.org

\(^{211}\) https://www.cs.uoregon.edu/research/tau/home.php

\(^{212}\) https://khuck.github.io/xpress-apex/

\(^{213}\) https://khuck.github.io/xpress-apex/

\(^{214}\) https://www.cmake.org

\(^{215}\) https://khuck.github.io/xpress-apex/

\(^{216}\) https://khuck.github.io/xpress-apex/usage/#hpx-louisiana-state-university

\(^{217}\) https://khuck.github.io/xpress-apex/
Static priority scheduling policy

- invoke using: `--hpx:queuing=static-priority (or -qs)`
- flag to turn on for build: `HPX_THREAD_SCHEDULERS=all` or `HPX_THREAD_SCHEDULERS=static-priority`

The static scheduling policy maintains one queue per OS thread from which each OS thread pulls its tasks (user threads). Threads are distributed in a round robin fashion. There is no thread stealing in this policy.

Local scheduling policy

- invoke using: `--hpx:queuing=local (or -ql)`
- flag to turn on for build: `HPX_THREAD_SCHEDULERS=all` or `HPX_THREAD_SCHEDULERS=local`

The local scheduling policy maintains one queue per OS thread from which each OS thread pulls its tasks (user threads).

Static scheduling policy

- invoke using: `--hpx:queuing=static`
- flag to turn on for build: `HPX_THREAD_SCHEDULERS=all` or `HPX_THREAD_SCHEDULERS=static`

The static scheduling policy maintains one queue per OS thread from which each OS thread pulls its tasks (user threads). Threads are distributed in a round robin fashion. There is no thread stealing in this policy.

Priority ABP scheduling policy

- invoke using: `--hpx:queuing=abp-priority-fifo`
- flag to turn on for build: `HPX_THREAD_SCHEDULERS=all` or `HPX_THREAD_SCHEDULERS=abp-priority`

Priority ABP policy maintains a double ended lock free queue for each OS thread. By default the number of high priority queues is equal to the number of OS threads; the number of high priority queues can be specified on the command line using `--hpx:high-priority-threads`. High priority threads are executed by the first OS threads before any other work is executed. When a queue is empty work will be taken from high priority queues first. There is one low priority queue from which threads will be scheduled only when there is no other work. For this scheduling policy there is an option to turn on NUMA sensitivity using the command line option `--hpx:numa-sensitive`. When NUMA sensitivity is turned on work stealing is done from queues associated with the same NUMA domain first, only after that work is stolen from other NUMA domains.

This scheduler can be used with two underlying queuing policies (FIFO: first-in-first-out, and LIFO: last-in-first-out). In order to use the LIFO policy use the command line option `--hpx:queuing=abp-priority-lifo`.

The HPX resource partitioner

The HPX resource partitioner lets you take the execution resources available on a system—processing units, cores, and numa domains—and assign them to thread pools. By default HPX creates a single thread pool name `default`. While this is good for most use cases, the resource partitioner lets you create multiple thread pools with custom resources and options.

Creating custom thread pools is useful for cases where you have tasks which absolutely need to run without interference from other tasks. An example of this is when using MPI\(^\text{218}\) for distribution instead of the built-in mechanisms in HPX (useful in legacy applications). In this case one can create a thread pool containing a single thread for MPI\(^\text{219}\).

communication. MPI tasks will then always run on the same thread, instead of potentially being stuck in a queue behind other threads.

Note that HPX thread pools are completely independent from each other in the sense that task stealing will never happen between different thread pools. However, tasks running on a particular thread pool can schedule tasks on another thread pool.

**Note:** It is simpler in some situations to schedule important tasks with high priority instead of using a separate thread pool.

### Using the resource partitioner

In order to create custom thread pools the resource partitioner needs to be set up before HPX is initialized by creating an instance of `hpx::resource::partitioner`:

```cpp
#include <hpx/hpx_init.hpp>
#include <hpx/resource_partitioner/partitioner.hpp>

int hpx_main(int argc, char* argv[])  
{  
    return hpx::finalize();  
}

int main(int argc, char** argv)  
{  
    hpx::resource::partitioner rp(argc, argv);  
    hpx::init();  
}
```

Note that we have to pass `argc` and `argv` to the resource partitioner to be able to parse thread binding options passed on the command line. You should pass the same arguments to the `hpx::resource::partitioner` constructor as you would to `hpx::init` or `hpx::start`. Running the above code will have the same effect as not initializing it at all, i.e. a default thread pool will be created with the type and number of threads specified on the command line.

The resource partitioner class is the interface to add thread pools to the HPX runtime and to assign resources to the thread pools.

To add a thread pool use the `hpx::resource::partitioner::create_thread_pool` method. If you simply want to use the default scheduler and scheduler options it is enough to call `rp.create_thread_pool("my-thread-pool").`

Then, to add resources to the thread pool you can use the `hpx::resource::partitioner::add_resource` method. The resource partitioner exposes the hardware topology retrieved using Portable Hardware Locality (HWLOC) and lets you iterate through the topology to add the wanted processing units to the thread pool. Below is an example of adding all processing units from the first NUMA domain to a custom thread pool, unless there is only one NUMA domain in which case we leave the first processing unit for the default thread pool:

```cpp
#include <hpx/hpx_init.hpp>
#include <hpx/resource_partitioner/partitioner.hpp>
#include <iostream>

int hpx_main(int argc, char* argv[])  
{  
    // Add resources to the thread pool
    hpx::resource::partitioner rp(argc, argv);
    hpx::init();
    // Add all processing units from the first NUMA domain
    for (auto& domain : hwloc::domains)  
    {  
        for (auto& node : domain.nodes)  
        {  
            // Add the processing unit to the thread pool
            rp.add_resource(node.name());
        }
    }
    // Create the thread pool
    hpx::partitioner::create_thread_pool("my-thread-pool");
    return hpx::finalize();
}
```

---

221 https://www.open-mpi.org/projects/hwloc/
{  
    return hpx::finalize();  
}

int main(int argc, char* argv[])
{
    hpx::resource::partitioner rp(argc, argv);
    rp.create_thread_pool("my-thread-pool");
    bool one_numa_domain = rp.numa_domains().size() == 1;
    bool skipped_first_pu = false;
    hpx::resource::numa_domain const& d = rp.numa_domains()[0];
    for (const hpx::resource::core& c : d.cores())
    {
        for (const hpx::resource::pu& p : c.pus())
        {
            if (one_numa_domain && !skipped_first_pu)
            {
                skipped_first_pu = true;
                continue;
            }
            rp.add_resource(p, "my-thread-pool");
        }
    }
    hpx::init();
}

**Note:** Whatever processing units not assigned to a thread pool by the time `hpx::init` is called will be added to the default thread pool. It is also possible to explicitly add processing units to the default thread pool, and to create the default thread pool manually (in order to e.g. set the scheduler type).

**Tip:** The command line option `--hpx:print-bind` is useful for checking that the thread pools have been set up the way you expect.

### Advanced usage

It is possible to customize the built in schedulers by passing scheduler options to `hpx::resource::partitioner::create_thread_pool`. It is also possible to create and use custom schedulers.

**Note:** It is not recommended to create your own scheduler. The HPX developers use this to experiment with new scheduler designs before making them available to users via the standard mechanisms of choosing a scheduler (command line options). If you would like to experiment with a custom scheduler the resource partitioner example `shared_priority_queue_scheduler.cpp` contains a fully implemented scheduler with logging etc. to make exploration easier.
To choose a scheduler and custom mode for a thread pool, pass additional options when creating the thread pool like this:

```cpp
rp.create_thread_pool("my-thread-pool",
    hpx::resource::policies::local_priority_lifo,
    hpx::policies::scheduler_mode(
        hpx::policies::scheduler_mode::default |
        hpx::policies::scheduler_mode::enable_elasticity));
```

The available schedulers are documented here: `hpx::resource::scheduling_policy`, and the available scheduler modes here: `hpx::threads::policies::scheduler_mode`. Also see the examples folder for examples of advanced resource partitioner usage: `simple_resource_partitioner.cpp` and `oversubscribing_resource_partitioner.cpp`.

## 2.5.12 Miscellaneous

### Error handling

Like in any other asynchronous invocation scheme it is important to be able to handle error conditions occurring while the asynchronous (and possibly remote) operation is executed. In HPX all error handling is based on standard C++ exception handling. Any exception thrown during the execution of an asynchronous operation will be transferred back to the original invocation locality, where it is rethrown during synchronization with the calling thread.

The source code for this example can be found here: `error_handling.cpp`.

### Working with exceptions

For the following description we assume that the function `raise_exception()` is executed by invoking the plain action `raise_exception_type`.

```cpp
// error_handling_raise_exception
void raise_exception()
{
    HPX_THROW_EXCEPTION(hpx::no_success, "raise_exception", "simulated error");
}
```

The exception is thrown using the macro `HPX_THROW_EXCEPTION`. The type of the thrown exception is `hpx::exception`. This associates additional diagnostic information with the exception, such as file name and line number, locality id and thread id, and stack backtrace from the point where the exception was thrown.

Any exception thrown during the execution of an action is transferred back to the (asynchronous) invocation site. It will be rethrown in this context when the calling thread tries to wait for the result of the action by invoking either `future<>::get()` or the synchronous action invocation wrapper as shown here:

```cpp
// Error reporting using exceptions
// exception_diagnostic_information
hpx::cout << "Error reporting using exceptions\n";
try {
    // invoke raise_exception() which throws an exception
    raise_exception_action do_it;
    do_it(hpx::find_here());
} catch (hpx::exception const& e) {
    // Print just the essential error information.
    hpx::cout << "caught exception: ";
    hpx::cout << e.what() << "\n\n";
    // Print all of the available diagnostic information as stored with the exception.
}
```

(continues on next page)
hpx::cout << "diagnostic information:"
<< hpx::diagnostic_information(e) << "\n";

Note: The exception is transferred back to the invocation site even if it is executed on a different locality.

Additionally, this example demonstrates how an exception thrown by an (possibly remote) action can be handled. It shows the use of hpx::diagnostic_information which retrieves all available diagnostic information from the exception as a formatted string. This includes, for instance, the name of the source file and line number, the sequence number of the OS-thread and the HPX-thread id, the locality id and the stack backtrace of the point where the original exception was thrown.

Under certain circumstances it is desirable to output only some of the diagnostics, or to output those using different formatting. For this case, HPX exposes a set of lower level functions as demonstrated in the following code snippet:

```cpp
// Detailed error reporting using exceptions
// [exception_diagnostic_elements
hpx::cout << "Detailed error reporting using exceptions\n";
try {
    // Invoke raise_exception() which throws an exception.
    raise_exception_action do_it;
    do_it(hpx::find_here());
} catch (hpx::exception const & e) {
    // Print the elements of the diagnostic information separately.
    hpx::cout << "{what}: " << hpx::get_error_what(e) << "\n";
    hpx::cout << "{locality-id}: " << hpx::get_error_locality_id(e) << "\n";
    hpx::cout << "{hostname}: " << hpx::get_error_host_name(e) << "\n";
    hpx::cout << "{pid}: " << hpx::get_error_process_id(e) << "\n";
    hpx::cout << "{function}: " << hpx::get_error_function_name(e) << "\n";
    hpx::cout << "{file}: " << hpx::get_error_file_name(e) << "\n";
    hpx::cout << "{line}: " << hpx::get_error_line_number(e) << "\n";
    hpx::cout << "{os-thread}: " << hpx::get_error_os_thread(e) << "\n";
    hpx::cout << "{thread-id}: " << std::hex << hpx::get_error_thread_id(e) << "\n";
    hpx::cout << "{thread-description}: " << hpx::get_error_thread_description(e) << "\n";
    hpx::cout << "{state}: " << std::hex << hpx::get_error_state(e) << "\n";
    hpx::cout << "{stack-trace}: " << hpx::get_error_backtrace(e) << "\n";
    hpx::cout << "{env}: " << hpx::get_error_env(e) << "\n";
}
```

Working with error codes

Most of the API functions exposed by HPX can be invoked in two different modes. By default those will throw an exception on error as described above. However, sometimes it is desirable not to throw an exception in case of an error condition. In this case an object instance of the hpx::error_code type can be passed as the last argument to the API function. In case of an error the error condition will be returned in that hpx::error_code instance. The following example demonstrates extracting the full diagnostic information without exception handling:

```cpp
{
    // [error_handling_diagnostic_information
    hpx::cout << "Error reporting using error code\n";
    // Create a new error_code instance.
```
hpx::error_code ec;

// If an instance of an error_code is passed as the last argument while
// invoking the action, the function will not throw in case of an error
// but store the error information in this error_code instance instead.
raise_exception_action do_it;
do_it(hpx::find_here(), ec);

if (ec) {
    // Print just the essential error information.
hpx::cout << "returned error: " << ec.get_message() << "\n";

    // Print all of the available diagnostic information as stored with
    // the exception.
hpx::cout << "diagnostic information:"
        << hpx::diagnostic_information(ec) << "\n";
}

Note: The error information is transferred back to the invocation site even if it is executed on a different locality.

This example show how an error can be handled without having to resolve to exceptions and that the returned hpx::error_code instance can be used in a very similar way as the hpx::exception type above. Simply pass it to the hpx::diagnostic_information which retrieves all available diagnostic information from the error code instance as a formatted string.

As for handling exceptions, when working with error codes, under certain circumstances it is desirable to output only some of the diagnostics, or to output those using different formatting. For this case, HPX exposes a set of lower level functions usable with error codes as demonstrated in the following code snippet:

{  
    // Print the elements of the diagnostic information separately.
hpx::cout << "\n";

    hpx::cout << "\n";

    hpx::cout << "\n";

    hpx::cout << "\n";

(continues on next page)
Lightweight error codes

Sometimes it is not desirable to collect all the ambient information about the error at the point where it happened as this might impose too much overhead for simple scenarios. In this case, HPX provides a lightweight error code facility which will hold the error code only. The following snippet demonstrates its use:

```cpp
{
    // Error reporting using an lightweight error code
    hpx::cout << "Error reporting using an lightweight error code\n"
     "\n"
    // Create a new error_code instance.
    hpx::error_code ec(hpx::lightweight);
    // If an instance of an error_code is passed as the last argument while
    // invoking the action, the function will not throw in case of an error
    // but store the error information in this error_code instance instead.
    raise_exception_action do_it;
    do_it(hpx::find_here(), ec);

    if (ec) {
        // Print just the essential error information.
        hpx::cout << "returned error: " << ec.get_message() << "\n"
        // Print all of the available diagnostic information as stored with
        // the exception.
        hpx::cout << "error code: " << ec.value() << "\n";
    }
```

All functions which retrieve other diagnostic elements from the `hpx::error_code` will fail if called with a lightweight error_code instance.

Utilities in HPX

In order to ease the burden of programming in HPX we have provided several utilities to users. The following section documents those facilities.
The HPX I/O-streams component

The HPX I/O-streams subsystem extends the standard C++ output streams `std::cout` and `std::cerr` to work in the distributed setting of an HPX application. All of the output streamed to `hpx::cout` will be dispatched to `std::cout` on the console locality. Likewise, all output generated from `hpx::cerr` will be dispatched to `std::cerr` on the console locality.

**Note:** All existing standard manipulators can be used in conjunction with `hpx::cout` and `hpx::cerr` Historically, HPX also defines `hpx::endl` and `hpx::flush` but those are just aliases for the corresponding standard manipulators.

In order to use either `hpx::cout` or `hpx::cerr` application codes need to `#include <hpx/include/iostreams.hpp>`. For an example, please see the simplest possible ‘Hello world’ program as included as an example with HPX:

```cpp
#include <hpx/include/iostreams.hpp>

int main()
{
    // Say hello to the world!
    hpx::cout << "Hello World!\n" << hpx::flush;
    return 0;
}
```

Additionally those applications need to link with the iostreams component. When using cmake this can be achieved by using the `COMPONENT_DEPENDENCIES` parameter, for instance:

```cmake
include(HPX_AddExecutable)

add_hpx_executable(
    hello_world
    SOURCES hello_world.cpp
    COMPONENT_DEPENDENCIES iostreams
)
```

**Note:** The `hpx::cout` and `hpx::cerr` streams buffer all output locally until a `std::endl` or `std::flush` is encountered. That means that no output will appear on the console as long as either of those is explicitly used.
2.5.13 Troubleshooting

This section contains commonly encountered problems when compiling or using HPX.

Undefined reference to boost::program_options

Boost::ProgramOptions is not ABI compatible between all C++ versions and compilers. Because of this you may see linker errors similar to this:

```plaintext
...: undefined reference to `boost::program_options::operator<<(std::ostream&,
   \_boost::program_options::options_description const&)
```

if you are not linking to a compatible version of Boost::ProgramOptions. We recommend that you use hpx::program_options, which is part of HPX, as a replacement for boost::program_options (see program_options). Until you have migrated to use hpx::program_options we recommend that you always build Boost libraries and HPX with the same compiler and C++ standard.

Undefined reference to hpx::cout

You may see an linker error message that looks a bit like this:

```plaintext
hello_world.cpp:(.text+0x5aa): undefined reference to `hpx::cout'
hello_world.cpp:(.text+0x5c3): undefined reference to `hpx::iostreams::flush'
```

This usually happens if you are trying to use HPX iostreams functionality such as hpx::cout but are not linking against it. The iostreams functionality is not part of the core HPX library, and must be linked to explicitly. Typically this can be solved by adding COMPONENT_DEPENDENCIES iostreams to a call to add_hpx_library/add_hpx_executable/hpx_setup_target if using CMake. See Creating HPX projects for more details.

2.6 Additional material

- 2-day workshop held at CSCS in 2016
  - Recorded lectures
  - Slides
- Tutorials repository
- STEllAR Group blog posts

2.7 Overview

HPX is organized into different sub-libraries. Those libraries can be seen as independent modules, with clear dependencies and no cycles. As an end-user, the use of these modules is completely transparent. If you use e.g. add_hpx_executable to create a target in your project you will automatically get all modules as dependencies. See All modules for a list of the available modules.

---

222 https://www.boost.org/
223 https://www.youtube.com/playlist?list=PL1tk5lGm7zvS7s-sqOOnmJQfFNjKze18
225 https://github.com/STEllAR-GROUP/tutorials
226 http://stellar-group.org/blog/
2.8 All modules

2.8.1 affinity

2.8.2 algorithms

The algorithms module exposes the full set of algorithms defined by the C++ standard. There is also partial support for C++ ranges.

See the API reference of the module for more details.

2.8.3 allocator_support

This module provides utilities for allocators. It contains `hpx::util::internal_allocator` which directly forwards allocation calls to `jemalloc`. This utility is mainly useful on Windows.

See the API reference of the module for more details.

2.8.4 assertion

The assertion library implements the macros `HPX_ASSERT` and `HPX_ASSERT_MSG`. Those two macros can be used to implement assertions which are turned off during a release build.

By default, the location and function where the assert has been called from are displayed when the assertion fires. This behavior can be modified by using `hpx::assertion::set_assertion_handler`. When HPX initializes, it uses this function to specify a more elaborate assertion handler. If your application needs to customize this, it needs to do so before calling `hpx::hpx_init`, `hpx::hpx_main` or using the C-main wrappers.

See the API reference of the module for more details.

2.8.5 basic_execution

The basic execution module is the main entry point to implement parallel and concurrent operations. It is modeled after P0443 with some additions and implementations for the described concepts. Most notably, it provides an abstraction for execution resources, execution contexts and execution agents in such a way, that it provides customization points that those aforementioned concepts can be replaced and combined with ease.

For that purpose, three virtual base classes are provided to be able to provide implementations with different properties:

- **resource_base**: This is the abstraction for execution resources, that is for example CPU cores or an accelerator.

- **context_base**: An execution context uses execution resources and is able to spawn new execution agents, as new threads of executions on the available resources.

- **agent_base**: The execution agent represents the thread of execution, and can be used to yield, suspend, resume or abort a thread of execution.

2.8.6 batch_environments

2.8.7 cache

This module provides two cache data structures:

- `hpx::util::cache::local_cache`
- `hpx::util::cache::lru_cache`

See the API reference of the module for more details.

---

http://wg21.link/p0443
2.8.8 checkpoint

A common need of users is to periodically backup an application. This practice provides resiliency and potential restart points in code. We have developed the concept of a checkpoint to support this use case.

Found in hpx/util/checkpoint.hpp, checkpoints are defined as objects which hold a serialized version of an object or set of objects at a particular moment in time. This representation can be stored in memory for later use or it can be written to disk for storage and/or recovery at a later point. In order to create and fill this object with data we use a function called save_checkpoint. In code the function looks like this:

```c++
hpx::future<hpx::util::checkpoint> hpx::util::save_checkpoint(a, b, c, ...);
```

save_checkpoint takes arbitrary data containers such as int, double, float, vector, and future and serializes them into a newly created checkpoint object. This function returns a future to a checkpoint containing the data. Let us look a simple use case below:

```c++
using hpx::util::checkpoint;
using hpx::util::save_checkpoint;

std::vector<int> vec{1,2,3,4,5};
hpx::future<checkpoint> save_checkpoint(vec);
```

Once the future is ready the checkpoint object will contain the vector vec and its five elements.

It is also possible to modify the launch policy used by save_checkpoint. This is accomplished by passing a launch policy as the first argument. It is important to note that passing hpx::launch::sync will cause save_checkpoint to return a checkpoint instead of a future to a checkpoint. All other policies passed to save_checkpoint will return a future to a checkpoint.

Sometimes checkpoints must be declared before they are used. save_checkpoint allows users to move pre-created checkpoint s into the function as long as they are the first container passing into the function (In the case where a launch policy is used, the checkpoint will immediately follow the launch policy). An example of these features can be found below:

```c++
char character = 'd';
int integer = 10;
float flt = 10.01f;
bool boolean = true;
std::string str = "I am a string of characters";
std::vector<char> vec(str.begin(), str.end());
checkpoint archive;

// Test 1
// test basic functionality
hpx::shared_future<checkpoint> f_archive = save_checkpoint(
    std::move(archive), character, integer, flt, boolean, str, vec);
```

Now that we can create checkpoints we now must be able to restore the objects they contain into memory. This is accomplished by the function restore_checkpoint. This function takes a checkpoint and fills its data into the containers it is provided. It is important to remember that the containers must be ordered in the same way they were placed into the checkpoint. For clarity see the example below:

```c++
char character2;
int integer2;
float flt2;
bool boolean2;
std::string str2;
std::vector<char> vec2;
```

(continues on next page)
The core utility of `checkpoint` is in its ability to make certain data persistent. Often this means that the data is needed to be stored in an object, such as a file, for later use. For these cases we have provided two solutions: stream operator overloads and access iterators.

We have created the two stream overloads `operator<<` and `operator>>` to stream data out of and into `checkpoint`. You can see an example of the overloads in use below:

```cpp
double a9 = 1.0, b9 = 1.1, c9 = 1.2;
std::ofstream test_file_9("test_file_9.txt");
hpx::future<checkpoint> f_9 = save_checkpoint(a9, b9, c9);
test_file_9 << f_9.get();
test_file_9.close();

double a9_1, b9_1, c9_1;
std::ifstream test_file_9_1("test_file_9.txt");
checkpoint archive9;
test_file_9_1 >> archive9;
restore_checkpoint(archive9, a9_1, b9_1, c9_1);
```

This is the primary way to move data into and out of a `checkpoint`. It is important to note, however, that users should be cautious when using a stream operator to load data an another function to remove it (or vice versa). Both `operator<<` and `operator>>` rely on a `.write()` and a `.read()` function respectively. In order to know how much data to read from the `std::istream`, the `operator<<` will write the size of the `checkpoint` before writing the `checkpoint` data. Correspondingly, the `operator>>` will read the size of the stored data before reading the data into new instance of `checkpoint`. As long as the user employs the `operator<<` and `operator>>` to stream the data this detail can be ignored.

**Important:** Be careful when mixing `operator<<` and `operator>>` with other facilities to read and write to a `checkpoint`. `operator<<` writes an extra variable and `operator>>` reads this variable back separately. Used together the user will not encounter any issues and can safely ignore this detail.

Users may also move the data into and out of a `checkpoint` using the exposed `.begin()` and `.end()` iterators. An example of this use case is illustrated below:

```cpp
std::ofstream test_file_7("checkpoint_test_file.txt");
std::vector<float> vec7{1.02f, 1.03f, 1.04f, 1.05f};
hpx::future<checkpoint> fut_7 = save_checkpoint(vec7);
checkpoint archive7 = fut_7.get();
std::copy(archive7.begin(), // Write data to ofstream
    archive7.end(), // ie. the file
    std::ostream_iterator<char>(test_file_7));
test_file_7.close();

std::vector<float> vec7_1;
std::vector<char> char_vec;
std::ifstream test_file_7_1("checkpoint_test_file.txt");
if (test_file_7_1)
{
    test_file_7_1.seekg(0, test_file_7_1.end);
    auto length = test_file_7_1.tellg();
    `...`
```
Checkpointing components

`save_checkpoint` and `restore_checkpoint` are also able to store components inside checkpoints. This can be done in one of two ways. First a client of the component can be passed to `save_checkpoint`. When the user wishes to resurrect the component she can pass a client instance to `restore_checkpoint`.

This technique is demonstrated below:

```cpp
// Try to checkpoint and restore a component with a client
std::vector<int> vec{10, 10, 10, 10, 10};

// Create a component instance through client constructor
data_client D(hpx::find_here(), std::move(vec));
uint32_t f3 = save_checkpoint(D);

// Create a new client
data_client E;

// Restore server inside client instance
restore_checkpoint(f3.get(), E);
```

The second way a user can save a component is by passing a `shared_ptr` to the component to `save_checkpoint`. This component can be resurrected by creating a new instance of the component type and passing a `shared_ptr` to the new instance to `restore_checkpoint`.

This technique is demonstrated below:

```cpp
// test checkpoint a component using a shared_ptr
std::vector<int> vec{1, 2, 3, 4, 5};
data_client A(hpx::find_here(), std::move(vec));

// Checkpoint Server
hpx::id_type old_id = A.get_id();

hpx::future<shared_ptr<data_server>> f_a_ptr = 
  hpx::get_ptr<data_server>(A.get_id());
std::shared_ptr<data_server> a_ptr = f_a_ptr.get();

hpx::future<checkpoint> f = save_checkpoint(a_ptr);

// Restore Server
// Create a new server instance
std::shared_ptr<data_server> b_server;
restore_checkpoint(f.get(), b_server);
```

### 2.8.9 collectives

The collectives module exposes a set of distributed collective operations. Those can be used to exchange data between participating sites in a coordinated way. At this point the module exposes the following collective primitives:
hpx::collectives::all_reduce: performs a reduction on data from each participating site to each participating site.

hpx::collectives::all_to_all: each participating site provides its element of the data to collect while all participating sites receive the data from every other site.

hpx::lcos::barrier: distributed barrier.

hpx::lcos::broadcast: performs a given action on all given global identifiers.

hpx::lcos::fold: performs a fold with a given action on all given global identifiers.

hpx::lcos::gather: gathers values from all participating sites.

hpx::lcos::latch: distributed latch.

hpx::lcos::reduce: performs a reduction on data from each participating site to a root site.

hpx::lcos::spmd_block: performs the same operation on a local image while providing handles to the other images.

See the API reference of the module for more details.

### 2.8.10 compute

The compute module provides utilities for handling task and memory affinity on host systems. The `compute_cuda` for extensions to CUDA programmable GPU devices.

See the API reference of the module for more details.

### 2.8.11 compute_cuda

This module extends the compute module to handle CUDA programmable GPU devices.

See the API reference of the module for more details.

### 2.8.12 concepts

This module provides helpers for emulating concepts. It provides the following macros:

- `HPX_CONCEPT_REQUIRES`
- `HPX_HAS_MEMBER_XXX_TRAIT_DEF`
- `HPX_HAS_XXX_TRAIT_DEF`

See the API reference of the module for more details.

### 2.8.13 concurrency

This module provides concurrency primitives useful for multi-threaded programming such as:

- `hpx::util::barrier`
- `hpx::util::cache_line_data` and `hpx::util::cache_aligned_data`: wrappers for aligning and padding data to cache lines.
- various lockfree queue data structures

See the API reference of the module for more details.
2.8.14 config

The config module contains various configuration options, typically hidden behind macros that choose the correct implementation based on the compiler and other available options.

See the API reference of the module for more details.

2.8.15 coroutines

The coroutines module provides coroutine (user-space thread) implementations for different platforms.

See the API reference of the module for more details.

2.8.16 datastructures

The datastructures module provides basic data structures (typically provided for compatibility with older C++ standards):

- `hpx::util::basic_any`
- `hpx::util::optional`
- `hpx::util::tuple`

See the API reference of the module for more details.

2.8.17 debugging

This module provides helpers for demangling symbol names.

See the API reference of the module for more details.

2.8.18 errors

This module provides support for exceptions and error codes:

- `hpx::exception`
- `hpx::error_code`
- `:e:enum:'hpx::error'`

See the API reference of the module for more details.

2.8.19 execution

This library implements executors and execution policies for use with parallel algorithms and other facilities related to managing the execution of tasks.

See the API reference of the module for more details.

2.8.20 filesystem

This module provides a compatibility layer for the C++17 filesystem library. If the filesystem library is available this module will simply forward its contents into the `hpx::filesystem` namespace. If the library is not available it will fall back to Boost.Filesystem instead.

See the API reference of the module for more details.
2.8.21 format

The format module exposes the `format` and `format_to` functions for formatting strings.
See the API reference of the module for more details.

2.8.22 functional

This module provides function wrappers and helpers for managing functions and their arguments.

- `hpx::util::function`
- `hpx::util::function_ref`
- `hpx::util::unique_function`
- `hpx::util::bind`
- `hpx::util::bind_back`
- `hpx::util::bind_front`
- `hpx::util::deferred_call`
- `hpx::util::invoke`
- `hpx::util::invoke_fused`
- `hpx::util::mem_fn`
- `hpx::util::one_shot`
- `hpx::util::protect`
- `hpx::util::result_of`

See the API reference of the module for more details.

2.8.23 hardware

The hardware module abstracts away hardware specific details of timestamps and CPU features.
See the API reference of the module for more details.

2.8.24 hashing

The hashing module provides two hashing implementations:

- `hpx::util::fibhash`
- `hpx::util::jenkins_hash`

See the API reference of the module for more details.

2.8.25 iterator_support

This module provides helpers for iterators. It provides `hpx::util::iterator_facade` and `hpx::util::iterator_adaptor` for creating new iterators, and the trait `hpx::util::is_iterator` along with more specific iterator traits.
See the API reference of the module for more details.
2.8.26 local_lcos
This module provides local LCOs which return futures:

- `hpx::lcos::local::and_gate`
- `hpx::lcos::local::channel`
- `hpx::lcos::local::one_element_channel`
- `hpx::lcos::local::receive_channel`
- `hpx::lcos::local::send_channel`
- `hpx::lcos::local::guard`
- `hpx::lcos::local::guard_set`
- `hpx::lcos::local::run_guarded`
- `hpx::lcos::local::conditional_trigger`
- `hpx::lcos::local::packaged_task`
- `hpx::lcos::local::promise`
- `hpx::lcos::local::receive_buffer`
- `hpx::lcos::local::trigger`

See the API reference of this module for more details.

2.8.27 logging
This module provides useful macros for logging information.
See the API reference of the module for more details.

2.8.28 memory
Part of this module is a forked version of boost::intrusive_ptr from Boost.SmartPtr.
See the API reference of the module for more details.

2.8.29 plugin
This module provides base utilities for creating plugins.
See the API reference of the module for more details.

2.8.30 preprocessor
This library contains useful preprocessor macros:

- `HPX_PP_CAT`
- `HPX_PP_EXPAND`
- `HPX_PP_NARGS`
- `HPX_PP_STRINGIZE`
- `HPX_PP_STRIP_PARENS`

See the API reference of the module for more details.
2.8.31 program_options

The module program_options is a direct fork of the Boost.ProgramOptions library (Boost V1.70.0). For more information about this library please see here\(^{228}\). In order to be included as an HPX module, the Boost.ProgramOptions library has been moved to the namespace `hpx::program_options`. We have also replaced all Boost facilities the library depends on with either the equivalent facilities from the standard library or from HPX. As a result, the HPX program_options module is fully interface compatible with Boost.ProgramOptions (sans the `hpx` namespace and the `#include <hpx/program_options.hpp>` changes that need to be applied to all code relying on this library.

All credit goes to Vladimir Prus, the author of the excellent Boost.ProgramOptions library. All bugs have been introduced by us.

See the API reference of the module for more details.

2.8.32 resiliency

In HPX, a program failure is a manifestation of a failing task. This module exposes several APIs that allow to manage failing tasks in a convenient way by either replaying a failed task or by replicating a specific task to begin with.

Task replay is analogous to the Checkpoint/Restart mechanism found in conventional execution models. The key difference being localized fault detection. When the runtime detects an error it replays the failing task as opposed to completely rolling back of the entire program to the previous checkpoint.

Task replication is designed to provide reliability enhancements by replicating a set of tasks and evaluating their results to determine a consensus among them. This technique is most effective in situations where there are few tasks in the critical path of the DAG which leaves the system underutilized or where hardware or software failures may result in an incorrect result instead of an error. However, the drawback of this method is the additional computational cost incurred by repeating a task multiple times.

The following API functions are exposed:

- `hpx::resiliency::async_replay`: This version of task replay will catch user defined exceptions and automatically reschedule the task N times before throwing an `hpx::resiliency::abort_replay_exception` if no task is able to complete execution without an exception.

- `hpx::resiliency::async_replay_validate`: This version of replay adds an argument to async replay which receives a user provided validation function to test the result of the task against. If the task’s output is validated, the result is returned. If the output fails the check or an exception is thrown, the task is replayed until no errors are encountered or the number of specified retries have been exceeded.

- `hpx::resiliency::async_replicate`: This is the most basic implementation of the task replication. The API returns the first result that runs without detecting any errors.

- `hpx::resiliency::async_replicate_validate`: This API additionally takes a validation function which evaluates the return values produced by the threads. The first task to compute a valid result is returned.

- `hpx::resiliency::async_replicate_vote`: This API adds a vote function to the basic replicate function. Many hardware or software failures are silent errors which do not interrupt program flow. In order to detect errors of this kind, it is necessary to run the task several times and compare the values returned by every version of the task. In order to determine which return value is “correct”, the API allow the user to provide a custom consensus function to properly form a consensus. This voting function then returns the “correct” answer.

- `hpx::resiliency::async_replicate_vote_validate`: This combines the features of the previously discussed replicate set. Replicate vote validate allows a user to provide a validation function to filter results. Additionally, as described in replicate vote, the user can provide a “voting function” which returns the consensus formed by the voting logic.

\(^{228}\) https://www.boost.org/doc/libs/1_70_0/doc/html/program_options.html

2.8. All modules
• `hpx::resiliency::dataflow_replay`: This version of dataflow replay will catch user defined exceptions and automatically reschedules the task N times before throwing an `hpx::resiliency::abort_replay_exception` if no task is able to complete execution without an exception. Any arguments for the executed task that are futures will cause the task invocation to be delayed until all of those futures have become ready.

• `hpx::resiliency::dataflow_replay_validate`: This version of replay adds an argument to dataflow replay which receives a user provided validation function to test the result of the task against. If the task’s output is validated, the result is returned. If the output fails the check or an exception is thrown, the task is replayed until no errors are encountered or the number of specified retries have been exceeded. Any arguments for the executed task that are futures will cause the task invocation to be delayed until all of those futures have become ready.

• `hpx::resiliency::dataflow_replicate`: This is the most basic implementation of the task replication. The API returns the first result that runs without detecting any errors. Any arguments for the executed task that are futures will cause the task invocation to be delayed until all of those futures have become ready.

• `hpx::resiliency::dataflow_replicate_validate`: This API additionally takes a validation function which evaluates the return values produced by the threads. The first task to compute a valid result is returned. Any arguments for the executed task that are futures will cause the task invocation to be delayed until all of those futures have become ready.

• `hpx::resiliency::dataflow_replicate_vote`: This API adds a vote function to the basic replicate function. Many hardware or software failures are silent errors which do not interrupt program flow. In order to detect errors of this kind, it is necessary to run the task several times and compare the values returned by every version of the task. In order to determine which return value is “correct”, the API allows the user to provide a custom consensus function to properly form a consensus. This voting function then returns the “correct” answer. Any arguments for the executed task that are futures will cause the task invocation to be delayed until all of those futures have become ready.

• `hpx::resiliency::dataflow_replicate_vote_validate`: This combines the features of the previously discussed replicate set. Replicate vote validate allows a user to provide a validation function to filter results. Additionally, as described in replicate vote, the user can provide a “voting function” which returns the consensus formed by the voting logic. Any arguments for the executed task that are futures will cause the task invocation to be delayed until all of those futures have become ready.

See the API reference of the module for more details.

### 2.8.33 resource_partitioner

### 2.8.34 segmented_algorithms

Segmented algorithms extend the usual parallel algorithms (algorithms) by providing overloads that work with distributed containers, such as partitioned vectors.

See the API reference of the module for more details.

### 2.8.35 serialization

This module provides serialization primitives and support for many C++ standard library and Boost types.

See the API reference of the module for more details.

### 2.8.36 static_reinit

### 2.8.37 statistics

This module provide some statistics utilities like rolling min/max and histogram.

See the API reference of the module for more details.
2.8.38 synchronization

This module provides synchronization primitives which should be used rather than the C++ standard ones:

• `hpx::lcos::local::barrier`
• `hpx::lcos::local::condition_variable`
• `hpx::lcos::local::counting_semaphore`
• `hpx::lcos::local::event`
• `hpx::lcos::local::latch`
• `hpx::lcos::local::mutex`
• `hpx::lcos::local::no_mutex`
• `hpx::lcos::local::once_flag`
• `hpx::lcos::local::recursive_mutex`
• `hpx::lcos::local::shared_mutex`
• `hpx::lcos::local::sliding_semaphore`
• `hpx::lcos::local::spinlock (std::mutex compatible spinlock)`
• `hpx::lcos::local::spinlock_no_backoff (boost::mutex compatible spinlock)`
• `hpx::lcos::local::spinlock_pool`

See the API reference of this module for more details.

2.8.39 testing

The testing module contains useful macros for testing. The results of tests can be printed with `hpx::util::report_errors`. The following macros are provided:

• `HPX_TEST`
• `HPX_TEST_MSG`
• `HPX_TEST_EQ`
• `HPX_TEST_NEQ`
• `HPX_TEST_LT`
• `HPX_TEST_LTE`
• `HPX_TEST_RANGE`
• `HPX_TEST_EQ_MSG`
• `HPX_TEST_NEQ_MSG`
• `HPX_SANITY`
• `HPX_SANITY_MSG`
• `HPX_SANITY_EQ`
• `HPX_SANITY_NEQ`
• `HPX_SANITY_LT`
• `HPX_SANITY_LTE`
• `HPX_SANITY_RANGE`
• `HPX_SANITY_EQ_MSG`

See the API reference of the module for more details.

### 2.8.40 thread_support
This module provides miscellaneous utilities for threading and concurrency.

See the API reference of the module for more details.

### 2.8.41 thread_manager

### 2.8.42 timing
This module provides the timing utilities (clocks and timers).

See the API reference of the module for more details.

### 2.8.43 topology
This module provides the class `hpx::threads::topology` which represents the hardware resources available on a node. The class is a light wrapper around the Portable Hardware Locality (HWLOC) library. The `hpx::threads::cpu_mask` is a small companion class that represents a set of resources on a node.

See the API reference of the module for more details.

### 2.8.44 type_support
This module provides helper facilities related to types.

See the API reference of the module for more details.

### 2.8.45 util
The util module provides miscellaneous standalone utilities.

See the API reference of the module for more details.

### 2.9 API reference

#### 2.9.1 Main HPX library reference

```cpp
template <typename Action>
struct async_result

#include <colocating_distribution_policy.hpp>

Note This function is part of the invocation policy implemented by this class

Public Types

```template<>```
```cpp
using type = hpx::future<typename traits::promise_local_result<typename hpx::traits::extract_action<Action>::remote_result_type>::type>

```template <typename Action>
```cpp
struct async_result

#include <default_distribution_policy.hpp>

Note This function is part of the invocation policy implemented by this class

229 https://www.open-mpi.org/projects/hwloc/

Chapter 2. What’s so special about HPX?
Public Types

template<>
using type = hpx::future<typename traits::promise_local_result<typename hpx::traits::extract_action<Action>::remote_result_type>::type>

struct binpacking_distribution_policy
#include <binpacking_distribution_policy.hpp> This class specifies the parameters for a binpacking distribution policy to use for creating a given number of items on a given set of localities. The binpacking policy will distribute the new objects in a way such that each of the localities will equalize the number of overall objects of this type based on a given criteria (by default this criteria is the overall number of objects of this type).

Public Functions

binpacking_distribution_policy()  
Default-construct a new instance of a binpacking_distribution_policy. This policy will represent one locality (the local locality).

binpacking_distribution_policy operator() (std::vector<id_type> const &locs, char const *perf_counter_name = default_binpacking_counter_name) const
Create a new default_distribution policy representing the given set of localities.

Parameters
• locs: [in] The list of localities the new instance should represent
• perf_counter_name: [in] The name of the performance counter which should be used as the distribution criteria (by default the overall number of existing instances of the given component type will be used).

binpacking_distribution_policy operator() (std::vector<id_type> &&locs, char const *perf_counter_name = default_binpacking_counter_name) const
Create a new default_distribution policy representing the given set of localities.

Parameters
• locs: [in] The list of localities the new instance should represent
• perf_counter_name: [in] The name of the performance counter which should be used as the distribution criteria (by default the overall number of existing instances of the given component type will be used).

binpacking_distribution_policy operator() (id_type const &loc, char const *perf_counter_name = default_binpacking_counter_name) const
Create a new default_distribution policy representing the given locality.

Parameters
• loc: [in] The locality the new instance should represent
• perf_counter_name: [in] The name of the performance counter which should be used as the distribution criteria (by default the overall number of existing instances of the given component type will be used).

template <typename Component, typename ... Ts>

hpx::future<hpx::id_type> create (Ts&... vs) const  
Create one object on one of the localities associated by this policy instance.
Return A future holding the global address which represents the newly created object

Parameters

• vs: [in] The arguments which will be forwarded to the constructor of the new object.

template <typename Component, typename ... Ts>
    hpx::future<std::vector<bulk_locality_result>> bulk_create (std::size_t count, Ts&&... vs) const
Create multiple objects on the localities associated by this policy instance

Return A future holding the list of global addresses which represent the newly created objects

Parameters

• count: [in] The number of objects to create
• vs: [in] The arguments which will be forwarded to the constructors of the new objects.

std::string const &get_counter_name () const
Returns the name of the performance counter associated with this policy instance.

std::size_t get_num_localities () const
Returns the number of associated localities for this distribution policy

Note This function is part of the creation policy implemented by this class

struct colocating_distribution_policy
    #include <colocating_distribution_policy.hpp> This class specifies the parameters for a distribution policy to use for creating a given number of items on the locality where a given object is currently placed.

Public Functions

colocating_distribution_policy ()
Default-construct a new instance of a colocating_distribution_policy. This policy will represent the local locality.

colocating_distribution_policy operator() (id_type const &id) const
Create a new colocating_distribution_policy representing the locality where the given object os current located

Parameters

• id: [in] The global address of the object with which the new instances should be colocated on

template <typename Client, typename Stub>
    colocating_distribution_policy operator() (client_base<Client, Stub> const &client) const
Create a new colocating_distribution_policy representing the locality where the given object os current located

Parameters

• client: [in] The client side representation of the object with which the new instances should be colocated on

template <typename Component, typename ... Ts>
    hpx::future<hpx::id_type> create (Ts&&... vs) const
Create one object on the locality of the object this distribution policy instance is associated with
Note  This function is part of the placement policy implemented by this class

Return  A future holding the global address which represents the newly created object

Parameters
  - \( \text{vs} \): [in] The arguments which will be forwarded to the constructor of the new object.

```cpp
template <typename Component, typename ... Ts>
    hpx::future<std::vector<bulk_locality_result>> bulk_create (std::size_t count, Ts&&... vs) const
```

Create multiple objects colocated with the object represented by this policy instance.

Note  This function is part of the placement policy implemented by this class

Return  A future holding the list of global addresses which represent the newly created objects

Parameters
  - \( \text{count} \): [in] The number of objects to create
  - \( \text{vs} \): [in] The arguments which will be forwarded to the constructors of the new objects.

```cpp
template <typename Action, typename ... Ts>
    async_result<Action>::type async (launch policy, Ts&&... vs) const
```

```cpp
template <typename Action, typename Callback, typename ... Ts>
    async_result<Action>::type async_cb (launch policy, Callback &&cb, Ts&&... vs) const
```

Note  This function is part of the invocation policy implemented by this class

```cpp
template <typename Action, typename Continuation, typename ... Ts>
    bool apply (Continuation &&c, threads::thread_priority priority, Ts&&... vs) const
```

```cpp
template <typename Action, typename Continuation, typename Callback, typename ... Ts>
    bool apply_cb (Continuation &&c, threads::thread_priority priority, Callback &&cb, Ts&&... vs) const
```

Note  This function is part of the invocation policy implemented by this class

```cpp
std::size_t get_num_localities() const
```

Returns the number of associated localities for this distribution policy

Note  This function is part of the creation policy implemented by this class

```cpp
hpx::id_type get_next_target() const
```

Returns the locality which is anticipated to be used for the next async operation

```cpp
struct default_distribution_policy
```

```
#include <default_distribution_policy.hpp>
```

This class specifies the parameters for a simple distribution policy to use for creating (and evenly distributing) a given number of items on a given set of localities.
Public Functions

**default_distribution_policy**()
Default-construct a new instance of a default_distribution_policy. This policy will represent one locality (the local locality).

**default_distribution_policy operator()** (std::vector<id_type> const &locs) const
Create a new default_distribution policy representing the given set of localities.

Parameters
- **locs**: [in] The list of localities the new instance should represent

**default_distribution_policy operator()** (std::vector<id_type> &&locs) const
Create a new default_distribution policy representing the given set of localities.

Parameters
- **locs**: [in] The list of localities the new instance should represent

**default_distribution_policy operator()** (id_type const &loc) const
Create a new default_distribution policy representing the given locality.

Parameters
- **loc**: [in] The locality the new instance should represent

**template<typename Component, typename ... Ts> hpx::future<hpx::id_type> create(Ts&&... vs) const**
Create one object on one of the localities associated by this policy instance.

**template<typename Component, typename ... Ts> hpx::future<std::vector<bulk_locality_result>> bulk_create(std::size_t count, Ts&&... vs) const**
Create multiple objects on the localities associated by this policy instance.

**Note** This function is part of the placement policy implemented by this class.

**Return** A future holding the list of global addresses which represent the newly created objects.

Parameters
- **count**: [in] The number of objects to create
- **vs**: [in] The arguments which will be forwarded to the constructors of the new objects.

**template<typename Action, typename ... Ts> async_result<Action>::type async(launch policy, Ts&&... vs) const**

**template<typename Action, typename Callback, typename ... Ts> async_result<Action>::type async_cb(launch policy, Callback &cb, Ts&&... vs) const**

**Note** This function is part of the invocation policy implemented by this class.
template <typename Action, typename Continuation, typename ... Ts>
bool apply (Continuation &&c, threads::thread_priority priority, Ts&&... vs) const

Note This function is part of the invocation policy implemented by this class

template <typename Action, typename ... Ts>
bool apply (threads::thread_priority priority, Ts&&... vs) const

Note This function is part of the invocation policy implemented by this class

template <typename Action, typename Callback, typename ... Ts>
bool apply_cb (Continuation &&c, threads::thread_priority priority, Callback &&&cb, Ts&&... vs) const

Note This function is part of the invocation policy implemented by this class

template <typename Action, typename Callback, typename ... Ts>
bool apply_cb (threads::thread_priority priority, Callback &&&cb, Ts&&... vs) const

std::size_t get_num_localities () const
Returns the number of associated localities for this distribution policy

Note This function is part of the creation policy implemented by this class

hpx::id_type get_next_target () const
Returns the locality which is anticipated to be used for the next async operation

template <>
template<>

struct hash<hpx::threads::thread_id>
#include <thread_data_fwd.hpp>

Public Functions

std::size_t operator() (hpx::threads::thread_id const &v) const

struct launch : public detail::policy_holder<>
#include <launch_policy.hpp> Launch policies for hpx::async etc.

Public Functions

launch ()
Default constructor. This creates a launch policy representing all possible launch modes

Public Static Attributes

const detail::fork_policy fork
Predefined launch policy representing asynchronous execution. The new thread is executed in a preferred way

const detail::sync_policy sync
Predefined launch policy representing synchronous execution.

const detail::deferred_policy deferred
Predefined launch policy representing deferred execution.

const detail::apply_policy apply
Predefined launch policy representing fire and forget execution.

const detail::select_policy_generator select
Predefined launch policy representing delayed policy selection.
template <typename ExPolicy = parallel::execution::parallel_policy>
class task_block
{
    //include <task_block.hpp> The class task_block defines an interface for forking and joining parallel tasks. The define_task_block and define_task_block_restore_thread function templates create an object of type task_block and pass a reference to that object to a user-provided callable object.

    An object of class task_block cannot be constructed, destroyed, copied, or moved except by the implementation of the task region library. Taking the address of a task_block object via operator& or addressof is ill formed. The result of obtaining its address by any other means is unspecified.

    A task_block is active if it was created by the nearest enclosing task block, where “task block” refers to an invocation of define_task_block or define_task_block_restore_thread and “nearest enclosing” means the most recent invocation that has not yet completed. Code designated for execution in another thread by means other than the facilities in this section (e.g., using thread or async) are not enclosed in the task region and a task_block passed to (or captured by) such code is not active within that code. Performing any operation on a task_block that is not active results in undefined behavior.

    The task_block that is active before a specific call to the run member function is not active within the asynchronous function that invoked run. (The invoked function should not, therefore, capture the task_block from the surrounding block.)

    Example:
    define_task_block([&] (auto& tr) {
        tr.run([&] { // Error: tr is not active
            tr.run([&] { f(); });
        });
        define_task_block([&] (auto& tr) { // Nested task block
            tr.run(f);
            /// ...
        });
        /// ...
    });

    Template Parameters
    • ExPolicy: The execution policy an instance of a task_block was created with. This defaults to parallel_policy.

    Public Types
    typedef ExPolicy execution_policy
    Refers to the type of the execution policy used to create the task_block.

    Public Functions
    execution_policy const &get_execution_policy() const
    Return the execution policy instance used to create this task_block

    template <typename F, typename ... Ts>
    void run (F &f, Ts&&... ts)
    Causes the expression f() to be invoked asynchronously. The invocation of f is permitted to run on an unspecified thread in an unordered fashion relative to the sequence of operations following the call to run(f) (the continuation), or indeterminately sequenced within the same thread as the continuation.

    The call to run synchronizes with the invocation of f. The completion of f() synchronizes with the next invocation of wait on the same task_block or completion of the nearest enclosing task block (i.e., the define_task_block or define_task_block_restore_thread that created this task block).
Requires: F shall be MoveConstructible. The expression, (void)f(), shall be well-formed.

Precondition: this shall be the active task_block.

Postconditions: A call to run may return on a different thread than that on which it was called.

Note The call to run is sequenced before the continuation as if run returns on the same thread. The invocation of the user-supplied callable object f may be immediate or may be delayed until compute resources are available. run might or might not return before invocation of f completes.

Exceptions

• This: function may throw task_canceled_exception, as described in Exception Handling.

```cpp
template <typename Executor, typename F, typename ... Ts>
void run (Executor &exec, F &&f, Ts&&... ts)
```

Causes the expression f() to be invoked asynchronously using the given executor. The invocation of f is permitted to run on an unspecified thread associated with the given executor and in an unordered fashion relative to the sequence of operations following the call to run(exec, f) (the continuation), or indeterminately sequenced within the same thread as the continuation.

The call to run synchronizes with the invocation of f. The completion of f() synchronizes with the next invocation of wait on the same task_block or completion of the nearest enclosing task block (i.e., the define_task_block or define_task_block_restore_thread that created this task block).

Requires: Executor shall be a type modeling the Executor concept. F shall be MoveConstructible. The expression, (void)f(), shall be well-formed.

Precondition: this shall be the active task_block.

Postconditions: A call to run may return on a different thread than that on which it was called.

Note The call to run is sequenced before the continuation as if run returns on the same thread. The invocation of the user-supplied callable object f may be immediate or may be delayed until compute resources are available. run might or might not return before invocation of f completes.

Exceptions

• This: function may throw task_canceled_exception, as described in Exception Handling.

```cpp
void wait ()
```

Blocks until the tasks spawned using this task_block have finished.

Precondition: this shall be the active task_block.

Postcondition: All tasks spawned by the nearest enclosing task region have finished. A call to wait may return on a different thread than that on which it was called.

Note The call to wait is sequenced before the continuation as if wait returns on the same thread.

Exceptions

• This: function may throw task_canceled_exception, as described in Exception Handling.
ExPolicy &policy()

Returns a reference to the execution policy used to construct this object.

Precondition: this shall be the active task_block.

ExPolicy const &policy() const

Returns a reference to the execution policy used to construct this object.

Precondition: this shall be the active task_block.

**Private Members**

mutex_type mtx_

std::vector<hpx::future<void>> tasks_

parallel::exception_list errors_

threads::thread_id_type id_

ExPolicy policy_

class task_canceled_exception: public exception

#include <task_block.hpp> The class task_canceled_exception defines the type of objects thrown by task_block::run or task_block::wait if they detect that an exception is pending within the current parallel region.

**Public Functions**

task_canceled_exception()

class thread_pool_base: public manage.executor

#include <thread_pool_base.hpp> The base class used to manage a pool of OS threads.

**Public Functions**

virtual void suspend_processing_unit_direct(std::size_t virt_core, error_code &ec = throws) = 0

Suspends the given processing unit. Blocks until the processing unit has been suspended.

Parameters

- **virt_core**: [in] The processing unit on the the pool to be suspended. The processing units are indexed starting from 0.

virtual void resume_processing_unit_direct(std::size_t virt_core, error_code &ec = throws) = 0

Resumes the given processing unit. Blocks until the processing unit has been resumed.

Parameters

- **virt_core**: [in] The processing unit on the the pool to be resumed. The processing units are indexed starting from 0.

virtual void resume_direct(error_code &ec = throws) = 0

Resumes the thread pool. Blocks until all OS threads on the thread pool have been resumed.

Parameters
• ec: [in,out] this represents the error status on exit, if this is pre-initialized to \texttt{hpx::throws} the function will throw on error instead.

\textbf{virtual void} \texttt{suspend\_direct}(\texttt{error\_code & ec = throws}) = 0

Suspends the thread pool. Blocks until all OS threads on the thread pool have been suspended.

\textbf{Note} A thread pool cannot be suspended from an HPX thread running on the pool itself.

\textbf{Parameters}

• ec: [in,out] this represents the error status on exit, if this is pre-initialized to \texttt{hpx::throws} the function will throw on error instead.

\textbf{Exceptions}

• \texttt{hpx::exception}: if called from an HPX thread which is running on the pool itself.

\textbf{struct thread\_pool\_init\_parameters}

\texttt{#include <thread\_pool\_base.hpp>}

\textbf{Public Functions}

\textbf{thread\_pool\_init\_parameters}(\texttt{std::string const \& name, std::size\_t index, policies::scheduler\_mode mode, std::size\_t num\_threads, std::size\_t thread\_offset, hpx::threads::policies::callback\_notifier \& notifier, hpx::threads::policies::detail::affinity\_data const \& affinity\_data, hpx::threads::detail::network\_background\_callback\_type const \& network\_background\_callback = hpx::threads::detail::network\_background\_callback\_type(), std::size\_t max\_background\_threads = std::size\_t(-1), std::size\_t max\_idle\_loop\_count = HPX\_IDLE\_LOOP\_COUNT\_MAX, std::size\_t max\_busy\_loop\_count = HPX\_BUSY\_LOOP\_COUNT\_MAX)}

\textbf{Public Members}

\texttt{std::string const \& name\_}
\texttt{std::size\_t index\_}
\texttt{policies::scheduler\_mode mode\_}
\texttt{std::size\_t num\_threads\_}
\texttt{std::size\_t thread\_offset\_}
\texttt{hpx::threads::policies::callback\_notifier \& notifier\_}
\texttt{hpx::threads::policies::detail::affinity\_data const \& affinity\_data\_}
\texttt{hpx::threads::detail::network\_background\_callback\_type const \& network\_background\_callback\_}
\texttt{std::size\_t max\_background\_threads\_}
\texttt{std::size\_t max\_idle\_loop\_count\_}
\texttt{std::size\_t max\_busy\_loop\_count\_}
struct unwrap
   
   #include <unwrap.hpp> A helper function object for functionally invoking hpx::util::unwrap. For more information please refer to its documentation.

struct unwrap_all
   
   #include <unwrap.hpp> A helper function object for functionally invoking hpx::util::unwrap_all. For more information please refer to its documentation.

template <std::size_t Depth>
struct unwrap_n
   
   #include <unwrap.hpp> A helper function object for functionally invoking hpx::util::unwrap_n. For more information please refer to its documentation.

template <typename Sequence>
struct when_any_result
   
   #include <when_any.hpp> Result type for when_any, contains a sequence of futures and an index pointing to a ready future.

   Public Members

   std::size_t index
      The index of a future which has become ready.

   Sequence futures
      The sequence of futures as passed to hpx::when_any.

   template <typename Sequence>
struct when_some_result
   
   #include <when_some.hpp> Result type for when_some, contains a sequence of futures and indices pointing to ready futures.

   Public Members

   std::vector<std::size_t> indices
      List of indices of futures which became ready.

   Sequence futures
      The sequence of futures as passed to hpx::when_some.

namespace applier
   
   The namespace applier contains all definitions needed for the class hpx::applier::applier and its related functionality. This namespace is part of the HPX core module.

namespace hpx

   Typedefs

   typedef util::unique_function_nonser<void ()> shutdown_function_type
      The type of a function which is registered to be executed as a shutdown or pre-shutdown function.

   typedef util::unique_function_nonser<void ()> startup_function_type
      The type of a function which is registered to be executed as a startup or pre-startup function.

   Enums

   enum runtime_mode
      A HPX runtime can be executed in two different modes: console mode and worker mode.

      Values:
HPX Documentation, 1.4.1

runtime_mode_invalid = -1
The runtime is the console locality.

runtime_mode_worker = 1
The runtime is a worker locality.

runtime_mode_connect = 2
The runtime is a worker locality connecting late

runtime_mode_default = 3
The runtime mode will be determined based on the command line arguments

runtime_mode_last

Functions

int init (util::function_nonser<int> hpx::program_options::variables_map &vm
> const &f, hpx::program_options::options_description const &desc_cmdline, int argc, char **argv, std::vector<std::string> const &cfg, startup_function_type startup = startup_function_type(),
shutdown_function_type shutdown = shutdown_function_type(), hpx::runtime_mode mode = hpx::runtime_mode_default)Main entry point for launching the HPX runtime system.

This is the main entry point for any HPX application. This function (or one of its overloads below) should
be called from the users main() function. It will set up the HPX runtime environment and schedule the
function given by f as a HPX thread. This overload will not call hpx_main.

Return The function returns the value, which has been returned from the user supplied f.

Note If the parameter mode is not given (defaulted), the created runtime system instance will be executed
in console or worker mode depending on the command line arguments passed in argc/argv. Otherwise it will be executed as specified by the parameter mode.

Parameters

• f: [in] The function to be scheduled as an HPX thread. Usually this function represents the main
entry point of any HPX application. If f is nullptr the HPX runtime environment will be
started without invoking f.

• desc_cmdline: [in] This parameter may hold the description of additional command line arguments
understood by the application. These options will be prepended to the default command line options understood by hpx::init (see description below).

• argc: [in] The number of command line arguments passed in argv. This is usually the unchanged value as passed by the operating system (to main()).

• argv: [in] The command line arguments for this application, usually that is the value as passed
by the operating system (to main()).

• cfg: A list of configuration settings which will be added to the system configuration before the
runtime instance is run. Each of the entries in this list must have the format of a fully defined
key/value pair from an ini-file (for instance ‘hpx.component.enabled=1’)

• startup: [in] A function to be executed inside a HPX thread before f is called. If this parameter
is not given no function will be executed.

• shutdown: [in] A function to be executed inside an HPX thread while hpx::finalize is executed.
If this parameter is not given no function will be executed.

• mode: [in] The mode the created runtime environment should be initialized in. There
has to be exactly one locality in each HPX application which is executed in console
mode (hpx::runtime_mode_console), all other localities have to be run in worker mode (hpx::runtime_mode_worker). Normally this is set up automatically, but sometimes it is necessary to explicitly specify the mode.

```c
int init (int (*f)) hpx::program_options::variables_map &vm
, hpx::program_options::options_description const &desc_cmdline, int argc, char **argv,
startup_function_type startup = startup_function_type(), shutdown_function_type shutdown = shut-
down_function_type(), hpx::runtime_mode mode = hpx::runtime_mode_default)
```

Main entry point for launching the HPX runtime system.

This is the main entry point for any HPX application. This function (or one of its overloads below) should be called from the users main() function. It will set up the HPX runtime environment and schedule the function given by f as a HPX thread. This overload will not call hpx_main.

**Return** The function returns the value, which has been returned from the user supplied f.

**Note** If the parameter mode is not given (defaulted), the created runtime system instance will be executed in console or worker mode depending on the command line arguments passed in argc/argv. Otherwise it will be executed as specified by the parameter mode.

**Parameters**

- **f**: [in] The function to be scheduled as an HPX thread. Usually this function represents the main entry point of any HPX application. If f is nullptr the HPX runtime environment will be started without invoking f.
- **desc_cmdline**: [in] This parameter may hold the description of additional command line arguments understood by the application. These options will be prepended to the default command line options understood by hpx::init (see description below).
- **argc**: [in] The number of command line arguments passed in argv. This is usually the unchanged value as passed by the operating system (to main()).
- **argv**: [in] The command line arguments for this application, usually that is the value as passed by the operating system (to main()).
- **startup**: [in] A function to be executed inside a HPX thread before f is called. If this parameter is not given no function will be executed.
- **shutdown**: [in] A function to be executed inside an HPX thread while hpx::finalize is executed. If this parameter is not given no function will be executed.
- **mode**: [in] The mode the created runtime environment should be initialized in. There has to be exactly one locality in each HPX application which is executed in console mode (hpx::runtime_mode_console), all other localities have to be run in worker mode (hpx::runtime_mode_worker). Normally this is set up automatically, but sometimes it is necessary to explicitly specify the mode.

```c
int init (hpx::program_options::options_description const &desc_cmdline, int argc, char **argv,
startup_function_type startup = startup_function_type(), shutdown_function_type shutdown = shut-
down_function_type(), hpx::runtime_mode mode = hpx::runtime_mode_default)
```

Main entry point for launching the HPX runtime system.

This is a simplified main entry point, which can be used to set up the runtime for an HPX application (the runtime system will be set up in console mode or worker mode depending on the command line settings).

In console mode it will execute the user supplied function hpx_main, in worker mode it will execute an empty hpx_main.
The function returns the value, which has been returned from `hpx_main` (or 0 when executed in worker mode).

Note If the parameter `mode` is not given (defaulted), the created runtime system instance will be executed in console or worker mode depending on the command line arguments passed in `argc/argv`. Otherwise it will be executed as specified by the parameter `mode`.

Parameters

- `desc_cmdline`: [in] This parameter may hold the description of additional command line arguments understood by the application. These options will be prepended to the default command line options understood by `hpx::init` (see description below).
- `argc`: [in] The number of command line arguments passed in `argv`. This is usually the unchanged value as passed by the operating system (to `main()`).
- `argv`: [in] The command line arguments for this application, usually that is the value as passed by the operating system (to `main()`).
- `startup`: [in] A function to be executed inside a HPX thread before `f` is called. If this parameter is not given no function will be executed.
- `shutdown`: [in] A function to be executed inside an HPX thread while `hpx::finalize` is executed. If this parameter is not given no function will be executed.
- `mode`: [in] The mode the created runtime environment should be initialized in. There has to be exactly one locality in each HPX application which is executed in console mode (`hpx::runtime_mode_console`), all other localities have to be run in worker mode (`hpx::runtime_mode_worker`). Normally this is set up automatically, but sometimes it is necessary to explicitly specify the mode.

```cpp
int init(hpx::program_options::options_description const &desc_cmdline, int argc, char **argv,
          std::vector<std::string> const &cfg, startup_function_type startup = startup_function_type(),
          shutdown_function_type shutdown = shutdown_function_type(), hpx::runtime_mode mode =
          hpx::runtime_mode_default)
```

Main entry point for launching the HPX runtime system.

This is a simplified main entry point, which can be used to set up the runtime for an HPX application (the runtime system will be set up in console mode or worker mode depending on the command line settings).

In console mode it will execute the user supplied function `hpx_main`, in worker mode it will execute an empty `hpx_main`.

The function returns the value, which has been returned from `hpx_main` (or 0 when executed in worker mode).

Note If the parameter `mode` is not given (defaulted), the created runtime system instance will be executed in console or worker mode depending on the command line arguments passed in `argc/argv`. Otherwise it will be executed as specified by the parameter `mode`.

Parameters

- `desc_cmdline`: [in] This parameter may hold the description of additional command line arguments understood by the application. These options will be prepended to the default command line options understood by `hpx::init` (see description below).
- `argc`: [in] The number of command line arguments passed in `argv`. This is usually the unchanged value as passed by the operating system (to `main()`).
- `argv`: [in] The command line arguments for this application, usually that is the value as passed by the operating system (to `main()`).
• **cfg**: A list of configuration settings which will be added to the system configuration before the runtime instance is run. Each of the entries in this list must have the format of a fully defined key/value pair from an ini-file (for instance ‘hpx.component.enabled=1’)

• **startup**: [in] A function to be executed inside a HPX thread before \( f \) is called. If this parameter is not given no function will be executed.

• **shutdown**: [in] A function to be executed inside an HPX thread while \( hpx::finalize \) is executed. If this parameter is not given no function will be executed.

• **mode**: [in] The mode the created runtime environment should be initialized in. There has to be exactly one locality in each HPX application which is executed in console mode (\( hpx::runtime_mode_console \)), all other localities have to be run in worker mode (\( hpx::runtime_mode_worker \)). Normally this is set up automatically, but sometimes it is necessary to explicitly specify the mode.

```cpp
int init(int argc, char **argv, std::vector<std::string> const &cfg, hpx::runtime_mode mode = hpx::runtime_mode_default)
```

Main entry point for launching the HPX runtime system.

This is a simplified main entry point, which can be used to set up the runtime for an HPX application (the runtime system will be set up in console mode or worker mode depending on the command line settings).

In console mode it will execute the user supplied function \( \text{hpx\_main} \), in worker mode it will execute an empty \( \text{hpx\_main} \).

**Return** The function returns the value, which has been returned from \( \text{hpx\_main} \) (or 0 when executed in worker mode).

**Note** The created runtime system instance will be executed in console or worker mode depending on the command line arguments passed in \( \text{argc/argv} \).

**Parameters**

• **argc**: [in] The number of command line arguments passed in \( \text{argv} \). This is usually the unchanged value as passed by the operating system (to \( \text{main}() \)).

• **argv**: [in] The command line arguments for this application, usually that is the value as passed by the operating system (to \( \text{main}() \)).

• **cfg**: A list of configuration settings which will be added to the system configuration before the runtime instance is run. Each of the entries in this list must have the format of a fully defined key/value pair from an ini-file (for instance ‘hpx.component.enabled=1’)

• **mode**: [in] The mode the created runtime environment should be initialized in. There has to be exactly one locality in each HPX application which is executed in console mode (\( hpx::runtime_mode_console \)), all other localities have to be run in worker mode (\( hpx::runtime_mode_worker \)). Normally this is set up automatically, but sometimes it is necessary to explicitly specify the mode.

```cpp
int init(hpx::program_options::options_description const &desc_cmdline, int argc, char **argv, hpx::runtime_mode mode)
```

Main entry point for launching the HPX runtime system.

This is a simplified main entry point, which can be used to set up the runtime for an HPX application (the runtime system will be set up in console mode or worker mode depending on the command line settings).

In console mode it will execute the user supplied function \( \text{hpx\_main} \), in worker mode it will execute an empty \( \text{hpx\_main} \).
Return  The function returns the value, which has been returned from `hpx_main` (or 0 when executed in worker mode).

Note  If the parameter `mode` is `runtime_mode_default`, the created runtime system instance will be executed in console or worker mode depending on the command line arguments passed in `argc/argv`. Otherwise it will be executed as specified by the parameter `mode`.

Parameters

- `desc_cmdline`: [in] This parameter may hold the description of additional command line arguments understood by the application. These options will be prepended to the default command line options understood by `hpx::init` (see description below).

- `argc`: [in] The number of command line arguments passed in `argv`. This is usually the unchanged value as passed by the operating system (to `main()`).

- `argv`: [in] The command line arguments for this application, usually that is the value as passed by the operating system (to `main()`).

- `mode`: [in] The mode the created runtime environment should be initialized in. There has to be exactly one locality in each HPX application which is executed in console mode (`hpx::runtime_mode_console`), all other localities have to be run in worker mode (`hpx::runtime_mode_worker`). Normally this is set up automatically, but sometimes it is necessary to explicitly specify the mode.

```cpp
int init(hpx::program_options::options_description const &desc_cmdline, int argc, char **argv,
          std::vector<std::string> const &cfg, hpx::runtime_mode mode)
```

Main entry point for launching the HPX runtime system.

This is a simplified main entry point, which can be used to set up the runtime for an HPX application (the runtime system will be set up in console mode or worker mode depending on the command line settings).

In console mode it will execute the user supplied function `hpx_main`, in worker mode it will execute an empty `hpx_main`.

Return  The function returns the value, which has been returned from `hpx_main` (or 0 when executed in worker mode).

Note  If the parameter `mode` is `runtime_mode_default`, the created runtime system instance will be executed in console or worker mode depending on the command line arguments passed in `argc/argv`. Otherwise it will be executed as specified by the parameter `mode`.

Parameters

- `desc_cmdline`: [in] This parameter may hold the description of additional command line arguments understood by the application. These options will be prepended to the default command line options understood by `hpx::init` (see description below).

- `argc`: [in] The number of command line arguments passed in `argv`. This is usually the unchanged value as passed by the operating system (to `main()`).

- `argv`: [in] The command line arguments for this application, usually that is the value as passed by the operating system (to `main()`).

- `cfg`: A list of configuration settings which will be added to the system configuration before the runtime instance is run. Each of the entries in this list must have the format of a fully defined key/value pair from an ini-file (for instance `"hpx.component.enabled=1"`)

- `mode`: [in] The mode the created runtime environment should be initialized in. There has to be exactly one locality in each HPX application which is executed in console mode (`hpx::runtime_mode_console`), all other localities have to be run in worker mode.
(hpx::runtime_mode_worker). Normally this is set up automatically, but sometimes it is necessary to explicitly specify the mode.

```cpp
int init (std::string const &app_name, int argc = 0, char **argv = nullptr, hpx::runtime_mode mode = hpx::runtime_mode_default)
```

Main entry point for launching the HPX runtime system.

This is a simplified main entry point, which can be used to set up the runtime for an HPX application (the runtime system will be set up in console mode or worker mode depending on the command line settings).

**Return** The function returns the value, which has been returned from `hpx_main` (or 0 when executed in worker mode).

**Note** The created runtime system instance will be executed in console or worker mode depending on the command line arguments passed in `argc/argv`.

**Parameters**

- `app_name`: [in] The name of the application.
- `argc`: [in] The number of command line arguments passed in `argv`. This is usually the unchanged value as passed by the operating system (to `main()`).
- `argv`: [in] The command line arguments for this application, usually that is the value as passed by the operating system (to `main()`).
- `mode`: [in] The mode the created runtime environment should be initialized in. There has to be exactly one locality in each HPX application which is executed in console mode (hpx::runtime_mode_console), all other localities have to be run in worker mode (hpx::runtime_mode_worker). Normally this is set up automatically, but sometimes it is necessary to explicitly specify the mode.

```cpp
int init (int argc = 0, char **argv = nullptr, hpx::runtime_mode mode = hpx::runtime_mode_default)
```

Main entry point for launching the HPX runtime system.

This is a simplified main entry point, which can be used to set up the runtime for an HPX application (the runtime system will be set up in console mode or worker mode depending on the command line settings).

**Return** The function returns the value, which has been returned from `hpx_main` (or 0 when executed in worker mode).

**Note** The created runtime system instance will be executed in console or worker mode depending on the command line arguments passed in `argc/argv`. If no command line arguments are passed, console mode is assumed.

**Note** If no command line arguments are passed the HPX runtime system will not support any of the default command line options as described in the section ‘HPX Command Line Options’.

**Parameters**

- `argc`: [in] The number of command line arguments passed in `argv`. This is usually the unchanged value as passed by the operating system (to `main()`).
- `argv`: [in] The command line arguments for this application, usually that is the value as passed by the operating system (to `main()`).
- `mode`: [in] The mode the created runtime environment should be initialized in. There has to be exactly one locality in each HPX application which is executed in console mode (hpx::runtime_mode_console), all other localities have to be run in worker mode (hpx::runtime_mode_worker). Normally this is set up automatically, but sometimes it is necessary to explicitly specify the mode.
int \textbf{init} (\texttt{std::vector<\texttt{std::string}> const \& \textit{cfg}, \texttt{hpx::runtime_mode \textit{mode} = hpx::runtime_mode_default})

Main entry point for launching the HPX runtime system.

This is a simplified main entry point, which can be used to set up the runtime for an HPX application (the runtime system will be set up in console mode or worker mode depending on the command line settings).

\textbf{Return} The function returns the value, which has been returned from \texttt{hpx\_main} (or 0 when executed in worker mode).

\textbf{Note} The created runtime system instance will be executed in console or worker mode depending on the command line arguments passed in \texttt{argc/argv}. If not command line arguments are passed, console mode is assumed.

\textbf{Note} If no command line arguments are passed the HPX runtime system will not support any of the default command line options as described in the section ‘HPX Command Line Options’.

\textbf{Parameters}

- \textit{cfg:} A list of configuration settings which will be added to the system configuration before the runtime instance is run. Each of the entries in this list must have the format of a fully defined key/value pair from an ini-file (for instance ‘hpx.component.enabled=1’)

- \textit{mode:} [in] The mode the created runtime environment should be initialized in. There has to be exactly one locality in each HPX application which is executed in console mode (hpx::runtime_mode_console), all other localities have to be run in worker mode (hpx::runtime_mode_worker). Normally this is set up automatically, but sometimes it is necessary to explicitly specify the mode.

int \textbf{init} (\texttt{int (*)\textit{f}}) \texttt{hpx::program\_options::variables\_map \&\textit{vm}}

, \texttt{std::string const \& \textit{app\_name}}, \texttt{int \textit{argc}, char **\textit{argv}, \texttt{hpx::runtime_mode \textit{mode} = hpx::runtime_mode_default})

Main entry point for launching the HPX runtime system.

This is a simplified main entry point, which can be used to set up the runtime for an HPX application (the runtime system will be set up in console mode or worker mode depending on the command line settings). It will schedule the function given by \textit{f} as a HPX thread. This overload will not call \texttt{hpx\_main}.

\textbf{Return} The function returns the value, which has been returned from the user supplied function \textit{f}.

\textbf{Note} The created runtime system instance will be executed in console or worker mode depending on the command line arguments passed in \texttt{argc/argv}.

\textbf{Parameters}

- \textit{f:} [in] The function to be scheduled as an HPX thread. Usually this function represents the main entry point of any HPX application. If \textit{f} is \texttt{nullptr} the HPX runtime environment will be started without invoking \textit{f}.

- \textit{app\_name:} [in] The name of the application.

- \textit{argc:} [in] The number of command line arguments passed in \texttt{argv}. This is usually the unchanged value as passed by the operating system (to \texttt{main()}).

- \textit{argv:} [in] The command line arguments for this application, usually that is the value as passed by the operating system (to \texttt{main()}).

- \textit{mode:} [in] The mode the created runtime environment should be initialized in. There has to be exactly one locality in each HPX application which is executed in console mode (hpx::runtime_mode_console), all other localities have to be run in worker mode (hpx::runtime_mode_worker). Normally this is set up automatically, but sometimes it is necessary to explicitly specify the mode.
int init (int (*f))(hp::program_options::variables_map &vm,
               int argc, char **argv, hpx::runtime_mode mode = hpx::runtime_mode_default)
Main entry point for launching the HPX runtime system.

This is a simplified main entry point, which can be used to set up the runtime for an HPX application (the runtime system will be set up in console mode or worker mode depending on the command line settings). It will schedule the function given by f as a HPX thread. This overload will not call hpx_main.

Return The function returns the value, which has been returned from the user supplied function f.

Note The created runtime system instance will be executed in console or worker mode depending on the command line arguments passed in argc/argv.

Parameters

- f: [in] The function to be scheduled as an HPX thread. Usually this function represents the main entry point of any HPX application. If f is nullptr the HPX runtime environment will be started without invoking f.
- argc: [in] The number of command line arguments passed in argv. This is usually the unchanged value as passed by the operating system (to main()).
- argv: [in] The command line arguments for this application, usually that is the value as passed by the operating system (to main()).
- mode: [in] The mode the created runtime environment should be initialized in. There has to be exactly one locality in each HPX application which is executed in console mode (hpx::runtime_mode_console), all other localities have to be run in worker mode (hpx::runtime_mode_worker). Normally this is set up automatically, but sometimes it is necessary to explicitly specify the mode.

int init (util::function_nonser<int>) int, char **
> const &f, std::string const &app_name, int argc, char **argv, hpx::runtime_mode mode = hpx::runtime_mode_default)
Main entry point for launching the HPX runtime system.

This is a simplified main entry point, which can be used to set up the runtime for an HPX application (the runtime system will be set up in console mode or worker mode depending on the command line settings). It will schedule the function given by f as a HPX thread. This overload will not call hpx_main.

Return The function returns the value, which has been returned from the user supplied function f.

Note The created runtime system instance will be executed in console or worker mode depending on the command line arguments passed in argc/argv.

Parameters

- f: [in] The function to be scheduled as an HPX thread. Usually this function represents the main entry point of any HPX application. If f is nullptr the HPX runtime environment will be started without invoking f.
- app_name: [in] The name of the application.
- argc: [in] The number of command line arguments passed in argv. This is usually the unchanged value as passed by the operating system (to main()).
- argv: [in] The command line arguments for this application, usually that is the value as passed by the operating system (to main()).
- mode: [in] The mode the created runtime environment should be initialized in. There has to be exactly one locality in each HPX application which is executed in console mode (hpx::runtime_mode_console), all other localities have to be run in worker mode.
(hpx::runtime_mode_worker). Normally this is set up automatically, but sometimes it is necessary to explicitly specify the mode.

```cpp
int init (util::function_nonser<int> int, char **
> const &f, int argc, char **argv, hpx::runtime_mode mode = hpx::runtime_mode_default) Main entry point for launching the HPX runtime system.

This is a simplified main entry point, which can be used to set up the runtime for an HPX application (the runtime system will be set up in console mode or worker mode depending on the command line settings). It will schedule the function given by f as a HPX thread. This overload will not call hpx_main.

**Return** The function returns the value, which has been returned from the user supplied function f.

**Note** The created runtime system instance will be executed in console or worker mode depending on the command line arguments passed in argc/argv.

**Parameters**

- **f**: [in] The function to be scheduled as an HPX thread. Usually this function represents the main entry point of any HPX application. If f is nullptr the HPX runtime environment will be started without invoking f.
- **argc**: [in] The number of command line arguments passed in argv. This is usually the unchanged value as passed by the operating system (to main()).
- **argv**: [in] The command line arguments for this application, usually that is the value as passed by the operating system (to main()).
- **mode**: [in] The mode the created runtime environment should be initialized in. There has to be exactly one locality in each HPX application which is executed in console mode (hpx::runtime_mode_console), all other localities have to be run in worker mode (hpx::runtime_mode_worker). Normally this is set up automatically, but sometimes it is necessary to explicitly specify the mode.

```cpp
int init (util::function_nonser<int> int, char **
> const &f, int argc, char **argv, std::vector<std::string> const &cfg, hpx::runtime_mode mode = hpx::runtime_mode_default) Main entry point for launching the HPX runtime system.

This is a simplified main entry point, which can be used to set up the runtime for an HPX application (the runtime system will be set up in console mode or worker mode depending on the command line settings). It will schedule the function given by f as a HPX thread. This overload will not call hpx_main.

**Return** The function returns the value, which has been returned from the user supplied function f.

**Note** The created runtime system instance will be executed in console or worker mode depending on the command line arguments passed in argc/argv.

**Parameters**

- **f**: [in] The function to be scheduled as an HPX thread. Usually this function represents the main entry point of any HPX application. If f is nullptr the HPX runtime environment will be started without invoking f.
- **argc**: [in] The number of command line arguments passed in argv. This is usually the unchanged value as passed by the operating system (to main()).
- **argv**: [in] The command line arguments for this application, usually that is the value as passed by the operating system (to main()).
• **cfg**: A list of configuration settings which will be added to the system configuration before the runtime instance is run. Each of the entries in this list must have the format of a fully defined key/value pair from an ini-file (for instance ‘hpx.component.enabled=1’)

• **mode**: [in] The mode the created runtime environment should be initialized in. There has to be exactly one locality in each HPX application which is executed in console mode (hpx::runtime_mode_console), all other localities have to be run in worker mode (hpx::runtime_mode_worker). Normally this is set up automatically, but sometimes it is necessary to explicitly specify the mode.

```c
int init(const std::vector<std::string> &cfg, hpx::runtime_mode mode = hpx::runtime_mode_default)
```

This is a simplified main entry point, which can be used to set up the runtime for an HPX application (the runtime system will be set up in console mode or worker mode depending on the command line settings). It will schedule the function given by `f` as a HPX thread. This overload will not call `hpx_main`.

**Return** The function returns the value, which has been returned from the user supplied function `f`.

**Note** The created runtime system instance will be executed in console or worker mode depending on the configuration passed in `cfg`.

**Parameters**

- **f**: [in] The function to be scheduled as an HPX thread. Usually this function represents the main entry point of any HPX application. If `f` is `nullptr` the HPX runtime environment will be started without invoking `f`.
- **cfg**: A list of configuration settings which will be added to the system configuration before the runtime instance is run. Each of the entries in this list must have the format of a fully defined key/value pair from an ini-file (for instance ‘hpx.component.enabled=1’)
- **mode**: [in] The mode the created runtime environment should be initialized in. There has to be exactly one locality in each HPX application which is executed in console mode (hpx::runtime_mode_console), all other localities have to be run in worker mode (hpx::runtime_mode_worker). Normally this is set up automatically, but sometimes it is necessary to explicitly specify the mode.

```c
bool start(const std::vector<std::string> &cfg, hpx::program_options::variables_map &vm)
```

This is the main, non-blocking entry point for any HPX application. This function (or one of its overloads below) should be called from the users `main()` function. It will set up the HPX runtime environment and schedule the function given by `f` as a HPX thread. It will return immediately after that. Use `hpx::wait` and `hpx::stop` to synchronize with the runtime system’s execution. This overload will not call `hpx_main`.

**Return** The function returns `true` if command line processing succeeded and the runtime system was started successfully. It will return `false` otherwise.

**Note** If the parameter `mode` is not given (defaulted), the created runtime system instance will be executed in console or worker mode depending on the command line arguments passed in `argc/argv`. Otherwise it will be executed as specified by the parameter `mode`.
Parameters

- \( f \): [in] The function to be scheduled as an HPX thread. Usually this function represents the main entry point of any HPX application. If \( f \) is nullptr the HPX runtime environment will be started without invoking \( f \).

- desc_cmdline: [in] This parameter may hold the description of additional command line arguments understood by the application. These options will be prepended to the default command line options understood by hpx::init (see description below).

- argc: [in] The number of command line arguments passed in argv. This is usually the unchanged value as passed by the operating system (to main()).

- argv: [in] The command line arguments for this application, usually that is the value as passed by the operating system (to main()).

- cfg: A list of configuration settings which will be added to the system configuration before the runtime instance is run. Each of the entries in this list must have the format of a fully defined key/value pair from an ini-file (for instance ‘hpx.component.enabled=1’)

- startup: [in] A function to be executed inside a HPX thread before \( f \) is called. If this parameter is not given no function will be executed.

- shutdown: [in] A function to be executed inside an HPX thread while hpx::finalize is executed. If this parameter is not given no function will be executed.

- mode: [in] The mode the created runtime environment should be initialized in. There has to be exactly one locality in each HPX application which is executed in console mode (hpx::runtime_mode_console), all other localities have to be run in worker mode (hpx::runtime_mode_worker). Normally this is set up automatically, but sometimes it is necessary to explicitly specify the mode.

```cpp
bool start(int (*f)) hpx::program_options::variables_map &vm,
  hpx::program_options::options_description const &desc_cmdline,
  int argc, char **argv,
  startup_function_type startup = startup_function_type(),
  shutdown_function_type shutdown = shutdown_function_type(),
  hpx::runtime_mode mode = hpx::runtime_mode_default)
```

This is the main, non-blocking entry point for any HPX application. This function (or one of its overloads below) should be called from the users main() function. It will set up the HPX runtime environment and schedule the function given by \( f \) as a HPX thread. It will return immediately after that. Use hpx::wait and hpx::stop to synchronize with the runtime system’s execution. This overload will not call hpx_main.

Return The function returns true if command line processing succeeded and the runtime system was started successfully. It will return false otherwise.

Note If the parameter mode is not given (defaulted), the created runtime system instance will be executed in console or worker mode depending on the command line arguments passed in argc/argv. Otherwise it will be executed as specified by the parameter mode.

Parameters

- \( f \): [in] The function to be scheduled as an HPX thread. Usually this function represents the main entry point of any HPX application. If \( f \) is nullptr the HPX runtime environment will be started without invoking \( f \).

- desc_cmdline: [in] This parameter may hold the description of additional command line arguments understood by the application. These options will be prepended to the default command line options understood by hpx::init (see description below).
bool start (hpx::program_options::options_description const &desc_cmdline, int argc, char **argv, 
startup_function_type startup = startup_function_type(), shutdown_function_type shutdown 
= shutdown_function_type(), hpx::runtime_mode mode = hpx::runtime_mode_default)

Main non-blocking entry point for launching the HPX runtime system.

This is a simplified main, non-blocking entry point, which can be used to set up the runtime for an HPX application (the runtime system will be set up in console mode or worker mode depending on the command line settings). It will return immediately after that. Use hpx::wait and hpx::stop to synchronize with the runtime system’s execution.

In console mode it will execute the user supplied function hpx_main, in worker mode it will execute an empty hpx_main.

Return The function returns true if command line processing succeeded and the runtime system was started successfully. It will return false otherwise.

Note If the parameter mode is not given (defaulted), the created runtime system instance will be executed in console or worker mode depending on the command line arguments passed in argc/argv. Otherwise it will be executed as specified by the parameter mode.

Parameters

• desc_cmdline: [in] This parameter may hold the description of additional command line arguments understood by the application. These options will be prepended to the default command line options understood by hpx::init (see description below).

• argc: [in] The number of command line arguments passed in argv. This is usually the unchanged value as passed by the operating system (to main()).

• argv: [in] The command line arguments for this application, usually that is the value as passed by the operating system (to main()).

• startup: [in] A function to be executed inside a HPX thread before f is called. If this parameter is not given no function will be executed.

• shutdown: [in] A function to be executed inside an HPX thread while hpx::finalize is executed. If this parameter is not given no function will be executed.

• mode: [in] The mode the created runtime environment should be initialized in. There has to be exactly one locality in each HPX application which is executed in console mode (hpx::runtime_mode_console), all other localities have to be run in worker mode (hpx::runtime_mode_worker). Normally this is set up automatically, but sometimes it is necessary to explicitly specify the mode.
bool start (hpx::program_options::options_description const &desc_cmdline, int argc, char **argv, std::vector<std::string> const &cfg, startup_function_type startup = startup_function_type(), shutdown_function_type shutdown = shutdown_function_type(), hpx::runtime_mode mode = hpx::runtime_mode_default)

Main non-blocking entry point for launching the HPX runtime system.

This is a simplified main, non-blocking entry point, which can be used to set up the runtime for an HPX application (the runtime system will be set up in console mode or worker mode depending on the command line settings). It will return immediately after that. Use hpx::wait and hpx::stop to synchronize with the runtime system’s execution.

In console mode it will execute the user supplied function hpx_main, in worker mode it will execute an empty hpx_main.

Return The function returns true if command line processing succeeded and the runtime system was started successfully. It will return false otherwise.

Note If the parameter mode is not given (defaulted), the created runtime system instance will be executed in console or worker mode depending on the command line arguments passed in argc/argv. Otherwise it will be executed as specified by the parameter mode.

Parameters

- desc_cmdline: [in] This parameter may hold the description of additional command line arguments understood by the application. These options will be prepended to the default command line options understood by hpx::init (see description below).
- argc: [in] The number of command line arguments passed in argv. This is usually the unchanged value as passed by the operating system (to main()).
- argv: [in] The command line arguments for this application, usually that is the value as passed by the operating system (to main()).
- cfg: A list of configuration settings which will be added to the system configuration before the runtime instance is run. Each of the entries in this list must have the format of a fully defined key/value pair from an ini-file (for instance ‘hpx.component.enabled=1’)
- startup: [in] A function to be executed inside a HPX thread before f is called. If this parameter is not given no function will be executed.
- shutdown: [in] A function to be executed inside an HPX thread while hpx::finalize is executed. If this parameter is not given no function will be executed.
- mode: [in] The mode the created runtime environment should be initialized in. There has to be exactly one locality in each HPX application which is executed in console mode (hpx::runtime_mode_console), all other localities have to be run in worker mode (hpx::runtime_mode_worker). Normally this is set up automatically, but sometimes it is necessary to explicitly specify the mode.
Return  The function returns \textit{true} if command line processing succeeded and the runtime system was started successfully. It will return \textit{false} otherwise.

Note  If the parameter \texttt{mode} is \texttt{runtime\_mode\_default}, the created runtime system instance will be executed in console or worker mode depending on the command line arguments passed in \texttt{argc/argv}. Otherwise it will be executed as specified by the parameter \texttt{mode}.

Parameters

\begin{itemize}
\item \texttt{argc}: [in] The number of command line arguments passed in \texttt{argv}. This is usually the unchanged value as passed by the operating system (to \texttt{main()}).
\item \texttt{argv}: [in] The command line arguments for this application, usually that is the value as passed by the operating system (to \texttt{main()}).
\item \texttt{cfg}: A list of configuration settings which will be added to the system configuration before the runtime instance is run. Each of the entries in this list must have the format of a fully defined key/value pair from an ini-file (for instance ‘hpx.component.enabled=1’)
\item \texttt{mode}: [in] The mode the created runtime environment should be initialized in. There has to be exactly one locality in each HPX application which is executed in console mode (\texttt{hpx:runtime\_mode\_console}), all other localities have to be run in worker mode (\texttt{hpx:runtime\_mode\_worker}). Normally this is set up automatically, but sometimes it is necessary to explicitly specify the mode.
\end{itemize}

\begin{verbatim}
bool \textbf{start} (hpx::program\_options::options\_description \texttt{const \&desc\_cmdline}, int \texttt{argc}, char **\texttt{argv},
\texttt{hpx::runtime\_mode \texttt{mode}})
\end{verbatim}

Main non-blocking entry point for launching the HPX runtime system.

This is a simplified main, non-blocking entry point, which can be used to set up the runtime for an HPX application (the runtime system will be set up in console mode or worker mode depending on the command line settings). It will return immediately after that. Use \texttt{hpx::wait} and \texttt{hpx::stop} to synchronize with the runtime system’s execution.

In console mode it will execute the user supplied function \texttt{hpx\_main}, in worker mode it will execute an empty \texttt{hpx\_main}.

Return  The function returns \textit{true} if command line processing succeeded and the runtime system was started successfully. It will return \textit{false} otherwise.

Note  If the parameter \texttt{mode} is \texttt{runtime\_mode\_default}, the created runtime system instance will be executed in console or worker mode depending on the command line arguments passed in \texttt{argc/argv}. Otherwise it will be executed as specified by the parameter \texttt{mode}.

Parameters

\begin{itemize}
\item \texttt{desc\_cmdline}: [in] This parameter may hold the description of additional command line arguments understood by the application. These options will be prepended to the default command line options understood by \texttt{hpx: initialise} (see description below).
\item \texttt{argc}: [in] The number of command line arguments passed in \texttt{argv}. This is usually the unchanged value as passed by the operating system (to \texttt{main()}).
\item \texttt{argv}: [in] The command line arguments for this application, usually that is the value as passed by the operating system (to \texttt{main()}).
\item \texttt{mode}: [in] The mode the created runtime environment should be initialized in. There has to be exactly one locality in each HPX application which is executed in console mode (\texttt{hpx:runtime\_mode\_console}), all other localities have to be run in worker mode (\texttt{hpx:runtime\_mode\_worker}). Normally this is set up automatically, but sometimes it is necessary to explicitly specify the mode.
\end{itemize}
bool **start** (std::string const &app_name, int argc = 0, char **argv = nullptr, hpx::runtime_mode mode = hpx::runtime_mode_default)

Main non-blocking entry point for launching the HPX runtime system.

This is a simplified main, non-blocking entry point, which can be used to set up the runtime for an HPX application (the runtime system will be set up in console mode or worker mode depending on the command line settings). It will return immediately after that. Use hpx::wait and hpx::stop to synchronize with the runtime system’s execution.

In console mode it will execute the user supplied function hpx_main, in worker mode it will execute an empty hpx_main.

**Return** The function returns true if command line processing succeeded and the runtime system was started successfully. It will return false otherwise.

**Note** If the parameter mode is runtime_mode_default, the created runtime system instance will be executed in console or worker mode depending on the command line arguments passed in argc/argv. Otherwise it will be executed as specified by the parameter mode.

**Parameters**

- **app_name**: [in] The name of the application.
- **argc**: [in] The number of command line arguments passed in argv. This is usually the unchanged value as passed by the operating system (to main()).
- **argv**: [in] The command line arguments for this application, usually that is the value as passed by the operating system (to main()).
- **cfg**: A list of configuration settings which will be added to the system configuration before the runtime instance is run. Each of the entries in this list must have the format of a fully defined key/value pair from an ini-file (for instance ‘hpx.component.enabled=1’)
- **mode**: [in] The mode the created runtime environment should be initialized in. There has to be exactly one locality in each HPX application which is executed in console mode (hpx::runtime_mode_console), all other localities have to be run in worker mode (hpx::runtime_mode_worker). Normally this is set up automatically, but sometimes it is necessary to explicitly specify the mode.
• **argc**: [in] The number of command line arguments passed in argv. This is usually the unchanged value as passed by the operating system (to main()).

• **argv**: [in] The command line arguments for this application, usually that is the value as passed by the operating system (to main()).

• **mode**: [in] The mode the created runtime environment should be initialized in. There has to be exactly one locality in each HPX application which is executed in console mode (hpx::runtime_mode_console), all other localities have to be run in worker mode (hpx::runtime_mode_worker). Normally this is set up automatically, but sometimes it is necessary to explicitly specify the mode.

```cpp
bool start (int argc = 0, char **argv = nullptr, hpx::runtime_mode mode = hpx::runtime_mode_default)
```

Main non-blocking entry point for launching the HPX runtime system.

This is a simplified main, non-blocking entry point, which can be used to set up the runtime for an HPX application (the runtime system will be set up in console mode or worker mode depending on the command line settings). It will return immediately after that. Use hpx::wait and hpx::stop to synchronize with the runtime system’s execution.

**Return** The function returns `true` if command line processing succeeded and the runtime system was started successfully. It will return `false` otherwise.

**Note** The created runtime system instance will be executed in console or worker mode depending on the command line arguments passed in argc/argv. If no command line arguments are passed, console mode is assumed.

**Note** If no command line arguments are passed the HPX runtime system will not support any of the default command line options as described in the section ‘HPX Command Line Options’.

**Parameters**

• **argc**: [in] The number of command line arguments passed in argv. This is usually the unchanged value as passed by the operating system (to main()).

• **argv**: [in] The command line arguments for this application, usually that is the value as passed by the operating system (to main()).

• **mode**: [in] The mode the created runtime environment should be initialized in. There has to be exactly one locality in each HPX application which is executed in console mode (hpx::runtime_mode_console), all other localities have to be run in worker mode (hpx::runtime_mode_worker). Normally this is set up automatically, but sometimes it is necessary to explicitly specify the mode.

```cpp
bool start (std::vector<std::string> const &cfg, hpx::runtime_mode mode = hpx::runtime_mode_default)
```

Main non-blocking entry point for launching the HPX runtime system.

This is a simplified main, non-blocking entry point, which can be used to set up the runtime for an HPX application (the runtime system will be set up in console mode or worker mode depending on the command line settings). It will return immediately after that. Use hpx::wait and hpx::stop to synchronize with the runtime system’s execution.

**Return** The function returns `true` if command line processing succeeded and the runtime system was started successfully. It will return `false` otherwise.

**Note** The created runtime system instance will be executed in console or worker mode depending on the command line arguments passed in argc/argv. If no command line arguments are passed, console mode is assumed.
Note If no command line arguments are passed the HPX runtime system will not support any of the default command line options as described in the section ‘HPX Command Line Options’.

Parameters

- **cfg**: A list of configuration settings which will be added to the system configuration before the runtime instance is run. Each of the entries in this list must have the format of a fully defined key/value pair from an ini-file (for instance ‘hpx.component.enabled=1’)
- **mode**: [in] The mode the created runtime environment should be initialized in. There has to be exactly one locality in each HPX application which is executed in console mode (hpx::runtime_mode_console), all other localities have to be run in worker mode (hpx::runtime_mode_worker). Normally this is set up automatically, but sometimes it is necessary to explicitly specify the mode.

```cpp
bool start (int (*f)) hpx::program_options::variables_map &vm,
    std::string const &app_name, int argc, char **argv,
    hpx::runtime_mode mode = hpx::runtime_mode_default
```

Main non-blocking entry point for launching the HPX runtime system.

This is a simplified main, non-blocking entry point, which can be used to set up the runtime for an HPX application (the runtime system will be set up in console mode or worker mode depending on the command line settings). It will return immediately after that. Use `hpx::wait` and `hpx::stop` to synchronize with the runtime system’s execution. This overload will schedule the function given by `f` as a HPX thread. It will not call `hpx_main`.

**Return** The function returns `true` if command line processing succeeded and the runtime system was started successfully. It will return `false` otherwise.

**Note** The created runtime system instance will be executed in console or worker mode depending on the command line arguments passed in `argc/argv`.

Parameters

- **f**: [in] The function to be scheduled as an HPX thread. Usually this function represents the main entry point of any HPX application.
- **app_name**: [in] The name of the application.
- **argc**: [in] The number of command line arguments passed in `argv`. This is usually the unchanged value as passed by the operating system (to `main()`).
- **argv**: [in] The command line arguments for this application, usually that is the value as passed by the operating system (to `main()`).
- **mode**: [in] The mode the created runtime environment should be initialized in. There has to be exactly one locality in each HPX application which is executed in console mode (hpx::runtime_mode_console), all other localities have to be run in worker mode (hpx::runtime_mode_worker). Normally this is set up automatically, but sometimes it is necessary to explicitly specify the mode.

```cpp
bool start (util::function_nonser<int>) int, char **
```

> `const &f, std::string const &app_name, int argc, char **argv, hpx::runtime_mode mode = hpx::runtime_mode_default`

Main non-blocking entry point for launching the HPX runtime system.

This is a simplified main, non-blocking entry point, which can be used to set up the runtime for an HPX application (the runtime system will be set up in console mode or worker mode depending on the command line settings). It will return immediately after that. Use `hpx::wait` and `hpx::stop` to synchronize with the runtime system’s execution. This overload will schedule the function given by `f` as a HPX thread. It will not call `hpx_main`.
Return The function returns `true` if command line processing succeeded and the runtime system was started successfully. It will return `false` otherwise.

Note The created runtime system instance will be executed in console or worker mode depending on the command line arguments passed in `argc/argv`.

Parameters

- `f`: [in] The function to be scheduled as an HPX thread. Usually this function represents the main entry point of any HPX application. If `f` is `nullptr` the HPX runtime environment will be started without invoking `f`.

- `app_name`: [in] The name of the application.

- `argc`: [in] The number of command line arguments passed in `argv`. This is usually the unchanged value as passed by the operating system (to `main()`).

- `argv`: [in] The command line arguments for this application, usually that is the value as passed by the operating system (to `main()`).

- `mode`: [in] The mode the created runtime environment should be initialized in. There has to be exactly one locality in each HPX application which is executed in console mode (hpx::runtime_mode_console), all other localities have to be run in worker mode (hpx::runtime_mode_worker). Normally this is set up automatically, but sometimes it is necessary to explicitly specify the mode.

```cpp
bool start (int (*f)) hpx::program_options::variables_map &vm
    , int argc, char **argv, hpx::runtime_mode mode = hpx::runtime_mode_default
```

This is a simplified main, non-blocking entry point for launching the HPX runtime system.

Return The function returns `true` if command line processing succeeded and the runtime system was started successfully. It will return `false` otherwise.

Note The created runtime system instance will be executed in console or worker mode depending on the command line arguments passed in `argc/argv`.

Parameters

- `f`: [in] The function to be scheduled as an HPX thread. Usually this function represents the main entry point of any HPX application. If `f` is `nullptr` the HPX runtime environment will be started without invoking `f`.

- `argc`: [in] The number of command line arguments passed in `argv`. This is usually the unchanged value as passed by the operating system (to `main()`).

- `argv`: [in] The command line arguments for this application, usually that is the value as passed by the operating system (to `main()`).

- `mode`: [in] The mode the created runtime environment should be initialized in. There has to be exactly one locality in each HPX application which is executed in console mode (hpx::runtime_mode_console), all other localities have to be run in worker mode (hpx::runtime_mode_worker). Normally this is set up automatically, but sometimes it is necessary to explicitly specify the mode.
bool start (util::function_nonser<int> f, int argc, char **argv, hpx::runtime_mode mode = hpx::runtime_mode_default)

Main non-blocking entry point for launching the HPX runtime system.

This is a simplified main, non-blocking entry point, which can be used to set up the runtime for an HPX application (the runtime system will be set up in console mode or worker mode depending on the command line settings). It will return immediately after that. Use hpx::wait and hpx::stop to synchronize with the runtime system’s execution. This overload will schedule the function given by f as a HPX thread. It will not call hpx_main.

Return The function returns true if command line processing succeeded and the runtime system was started successfully. It will return false otherwise.

Note The created runtime system instance will be executed in console or worker mode depending on the command line arguments passed in argc/argv.

Parameters

• f: [in] The function to be scheduled as an HPX thread. Usually this function represents the main entry point of any HPX application. If f is nullptr the HPX runtime environment will be started without invoking f.

• argc: [in] The number of command line arguments passed in argv. This is usually the unchanged value as passed by the operating system (to main()).

• argv: [in] The command line arguments for this application, usually that is the value as passed by the operating system (to main()).

• mode: [in] The mode the created runtime environment should be initialized in. There has to be exactly one locality in each HPX application which is executed in console mode (hpx::runtime_mode_console), all other localities have to be run in worker mode (hpx::runtime_mode_worker). Normally this is set up automatically, but sometimes it is necessary to explicitly specify the mode.

bool start (util::function_nonser<int> f, int argc, char **argv, std::vector<std::string> &cfg, hpx::runtime_mode mode = hpx::runtime_mode_default)

This is a simplified main, non-blocking entry point, which can be used to set up the runtime for an HPX application (the runtime system will be set up in console mode or worker mode depending on the command line settings). It will return immediately after that. Use hpx::wait and hpx::stop to synchronize with the runtime system’s execution. This overload will schedule the function given by f as a HPX thread. It will not call hpx_main.

Return The function returns true if command line processing succeeded and the runtime system was started successfully. It will return false otherwise.

Note The created runtime system instance will be executed in console or worker mode depending on the command line arguments passed in argc/argv.

Parameters

• f: [in] The function to be scheduled as an HPX thread. Usually this function represents the main entry point of any HPX application. If f is nullptr the HPX runtime environment will be started without invoking f.

• argc: [in] The number of command line arguments passed in argv. This is usually the unchanged value as passed by the operating system (to main()).
• **argv**: [in] The command line arguments for this application, usually that is the value as passed by the operating system (to `main()`).

• **cfg**: A list of configuration settings which will be added to the system configuration before the runtime instance is run. Each of the entries in this list must have the format of a fully defined key/value pair from an ini-file (for instance ‘hpx.component.enabled=1’)

• **mode**: [in] The mode the created runtime environment should be initialized in. There has to be exactly one locality in each HPX application which is executed in console mode (`hpx::runtime_mode_console`), all other localities have to be run in worker mode (`hpx::runtime_mode_worker`). Normally this is set up automatically, but sometimes it is necessary to explicitly specify the mode.

### bool start (util::function_nonser<int>) int, char **

```cpp
> const &f, std::vector<std::string> const &cfg, hpx::runtime_mode mode = hpx::runtime_mode_default
```

Main non-blocking entry point for launching the HPX runtime system. This is a simplified main, non-blocking entry point, which can be used to set up the runtime for an HPX application (the runtime system will be set up in console mode or worker mode depending on the command line settings). It will return immediately after that. Use `hpx::wait` and `hpx::stop` to synchronize with the runtime system’s execution. This overload will schedule the function given by `f` as a HPX thread. It will not call `hpx_main`.

**Return** The function returns `true` if command line processing succeeded and the runtime system was started successfully. It will return `false` otherwise.

**Note** The created runtime system instance will be executed in console or worker mode depending on the configuration passed in `cfg`.

**Parameters**

- **f**: [in] The function to be scheduled as an HPX thread. Usually this function represents the main entry point of any HPX application. If `f` is `nullptr` the HPX runtime environment will be started without invoking `f`.

- **cfg**: A list of configuration settings which will be added to the system configuration before the runtime instance is run. Each of the entries in this list must have the format of a fully defined key/value pair from an ini-file (for instance ‘hpx.component.enabled=1’)

- **mode**: [in] The mode the created runtime environment should be initialized in. There has to be exactly one locality in each HPX application which is executed in console mode (`hpx::runtime_mode_console`), all other localities have to be run in worker mode (`hpx::runtime_mode_worker`). Normally this is set up automatically, but sometimes it is necessary to explicitly specify the mode.

### int finalize (double shutdown_timeout, double localwait = -1.0, error_code &ec = throws)

Main function to gracefully terminate the HPX runtime system.

The function `hpx::finalize` is the main way to (gracefully) exit any HPX application. It should be called from one locality only (usually the console) and it will notify all connected localities to finish execution. Only after all other localities have exited this function will return, allowing to exit the console locality as well.

During the execution of this function the runtime system will invoke all registered shutdown functions (see `hpx::init`) on all localities.

The default value (`-1.0`) will try to find a globally set timeout value (can be set as the configuration parameter `hpx.shutdown_timeout`), and if that is not set or `-1.0` as well, it will disable any timeout, each connected locality will wait for all existing HPX-threads to terminate.
Parameters

- **shutdown_timeout**: This parameter allows to specify a timeout (in microseconds), specifying how long any of the connected localities should wait for pending tasks to be executed. After this timeout, all suspended HPX-threads will be aborted. Note, that this function will not abort any running HPX-threads. In any case the shutdown will not proceed as long as there is at least one pending/running HPX-thread.

The default value (-1.0) will try to find a globally set wait time value (can be set as the configuration parameter "hpx.finalize_wait_time"), and if this is not set or -1.0 as well, it will disable any addition local wait time before proceeding.

Parameters

- **localwait**: This parameter allows to specify a local wait time (in microseconds) before the connected localities will be notified and the overall shutdown process starts.

This function will block and wait for all connected localities to exit before returning to the caller. It should be the last HPX-function called by any application.

**Return** This function will always return zero.

**Note** As long as `ec` is not pre-initialized to `hpx::throws` this function doesn’t throw but returns the result code using the parameter `ec`. Otherwise it throws an instance of `hpx::exception`.

Parameters

- **ec**: [in,out] this represents the error status on exit, if this is pre-initialized to `hpx::throws` the function will throw on error instead.

Using this function is an alternative to `hpx::disconnect`, these functions do not need to be called both.

```cpp
int finalize(error_code &ec = throws)
```

Main function to gracefully terminate the HPX runtime system.

The function `hpx::finalize` is the main way to (graciously) exit any HPX application. It should be called from one locality only (usually the console) and it will notify all connected localities to finish execution. Only after all other localities have exited this function will return, allowing to exit the console locality as well.

During the execution of this function the runtime system will invoke all registered shutdown functions (see `hpx::init`) on all localities.

This function will block and wait for all connected localities to exit before returning to the caller. It should be the last HPX-function called by any application.

**Return** This function will always return zero.

**Note** As long as `ec` is not pre-initialized to `hpx::throws` this function doesn’t throw but returns the result code using the parameter `ec`. Otherwise it throws an instance of `hpx::exception`.

Parameters

- **ec**: [in,out] this represents the error status on exit, if this is pre-initialized to `hpx::throws` the function will throw on error instead.

Using this function is an alternative to `hpx::disconnect`, these functions do not need to be called both.

```cpp
HPX_NORETURN void hpx::terminate()
```

Terminate any application non-gracefully.

The function `hpx::terminate` is the non-graceful way to exit any application immediately. It can be called from any locality and will terminate all localities currently used by the application.
Note This function will cause HPX to call \texttt{std::terminate()} on all localities associated with this application. If the function is called not from an HPX thread it will fail and return an error using the argument \texttt{ec}.

The function \texttt{hplox.disconnect} can be used to disconnect a locality from a running HPX application.

During the execution of this function the runtime system will invoke all registered shutdown functions (see \texttt{hplox.init}) on this locality. The default value (-1.0) will try to find a globally set timeout value (can be set as the configuration parameter \texttt{"hpx.shutdown.timeout"}), and if that is not set or -1.0 as well, it will disable any timeout, each connected locality will wait for all existing HPX-threads to terminate.

**Parameters**

- \texttt{shutdown.timeout}: This parameter allows to specify a timeout (in microseconds), specifying how long this locality should wait for pending tasks to be executed. After this timeout, all suspended HPX-threads will be aborted. Note, that this function will not abort any running HPX-threads. In any case the shutdown will not proceed as long as there is at least one pending/running HPX-thread.

The default value (-1.0) will try to find a globally set wait time value (can be set as the configuration parameter \texttt{hpx.finalize_wait_time}), and if this is not set or -1.0 as well, it will disable any addition local wait time before proceeding.

**Parameters**

- \texttt{localwait}: This parameter allows to specify a local wait time (in microseconds) before the connected localities will be notified and the overall shutdown process starts.

This function will block and wait for this locality to finish executing before returning to the caller. It should be the last HPX-function called by any locality being disconnected.

**Return** This function will always return zero.

**Note** As long as \texttt{ec} is not pre-initialized to \texttt{hplox::throws} this function doesn’t throw but returns the result code using the parameter \texttt{ec}. Otherwise it throws an instance of \texttt{hplox::exception}.

**Parameters**

- \texttt{ec}: [in,out] this represents the error status on exit, if this is pre-initialized to \texttt{hplox::throws} the function will throw on error instead.

The function \texttt{hplox.disconnect} can be used to disconnect a locality from a running HPX application.

During the execution of this function the runtime system will invoke all registered shutdown functions (see \texttt{hplox.init}) on this locality.

This function will block and wait for this locality to finish executing before returning to the caller. It should be the last HPX-function called by any locality being disconnected.

**Return** This function will always return zero.

**Note** As long as \texttt{ec} is not pre-initialized to \texttt{hplox::throws} this function doesn’t throw but returns the result code using the parameter \texttt{ec}. Otherwise it throws an instance of \texttt{hplox::exception}.

**Parameters**
• ec: [in,out] this represents the error status on exit, if this is pre-initialized to `hpx::throws` the function will throw on error instead.

```cpp
int stop (error_code &ec = throws)
Stop the runtime system.
```

This function will block and wait for this locality to finish executing before returning to the caller. It should be the last HPX-function called on every locality. This function should be used only if the runtime system was started using `hpx::start`.

**Return** The function returns the value, which has been returned from the user supplied main HPX function (usually `hpx_main`).

```cpp
int suspend (error_code &ec = throws)
Suspend the runtime system.
```

The function `hpx::suspend` is used to suspend the HPX runtime system. It can only be used when running HPX on a single locality. It will block waiting for all thread pools to be empty. This function only be called when the runtime is running, or already suspended in which case this function will do nothing.

**Return** This function will always return zero.

**Note** As long as `ec` is not pre-initialized to `hpx::throws` this function doesn’t throw but returns the result code using the parameter `ec`. Otherwise it throws an instance of `hpx::exception`.

**Parameters**

• ec: [in,out] this represents the error status on exit, if this is pre-initialized to `hpx::throws` the function will throw on error instead.

```cpp
int resume (error_code &ec = throws)
Resume the HPX runtime system.
```

The function `hpx::resume` is used to resume the HPX runtime system. It can only be used when running HPX on a single locality. It will block waiting for all thread pools to be resumed. This function only be called when the runtime suspended, or already running in which case this function will do nothing.

**Return** This function will always return zero.

**Note** As long as `ec` is not pre-initialized to `hpx::throws` this function doesn’t throw but returns the result code using the parameter `ec`. Otherwise it throws an instance of `hpx::exception`.

**Parameters**

• ec: [in,out] this represents the error status on exit, if this is pre-initialized to `hpx::throws` the function will throw on error instead.

```cpp
bool register_thread (runtime *rt, char const *name, error_code &ec = throws)
Register the current kernel thread with HPX, this should be done once for each external OS-thread intended to invoke HPX functionality. Calling this function more than once will silently fail.
```

```cpp
void unregister_thread (runtime *rt)
Unregister the thread from HPX, this should be done once in the end before the external thread exists.
```

```cpp
naming::gid_type const &get_locality ()
The function `get_locality` returns a reference to the locality prefix.
```

```cpp
std::size_t get_runtime_instance_number ()
The function `get_runtime_instance_number` returns a unique number associated with the runtime instance the current thread is running in.
```
bool register_on_exit (util::function_ser<void>)
  > const & Register a function to be called during system shutdown.

bool is_starting ()
  Test whether the runtime system is currently being started.

  This function returns whether the runtime system is currently being started or not, e.g. whether the current state of the runtime system is hpx::state_startup

  Note This function needs to be executed on a HPX-thread. It will return false otherwise.

bool tolerate_node_faults ()
  Test if HPX runs in fault-tolerant mode.

  This function returns whether the runtime system is running in fault-tolerant mode

bool is_running ()
  Test whether the runtime system is currently running.

  This function returns whether the runtime system is currently running or not, e.g. whether the current state of the runtime system is hpx::state_running

  Note This function needs to be executed on a HPX-thread. It will return false otherwise.

bool is_stopped ()
  Test whether the runtime system is currently stopped.

  This function returns whether the runtime system is currently stopped or not, e.g. whether the current state of the runtime system is hpx::state_stopped

  Note This function needs to be executed on a HPX-thread. It will return false otherwise.

bool is_stopped_or_shutting_down ()
  Test whether the runtime system is currently being shut down.

  This function returns whether the runtime system is currently being shut down or not, e.g. whether the current state of the runtime system is hpx::state_stopped or hpx::state_shutdown

  Note This function needs to be executed on a HPX-thread. It will return false otherwise.

std::size_t get_num_worker_threads ()
  Return the number of worker OS- threads used to execute HPX threads.

  This function returns the number of OS-threads used to execute HPX threads. If the function is called while no HPX runtime system is active, it will return zero.

std::uint64_t get_system_uptime ()
  Return the system uptime measure on the thread executing this call.

  This function returns the system uptime measured in nanoseconds for the thread executing this call. If the function is called while no HPX runtime system is active, it will return zero.

void start_active_counters (error_code & ec = throws)
  Start all active performance counters, optionally naming the section of code.

  Note As long as ec is not pre-initialized to hpx::throws this function doesn’t throw but returns the result code using the parameter ec. Otherwise it throws an instance of hpx::exception.
**Note** The active counters are those which have been specified on the command line while executing the application (see command line option –hpx:print-counter)

**Parameters**

- **ec**: [in,out] this represents the error status on exit, if this is pre-initialized to `hpx::throws` the function will throw on error instead.

```cpp
void reset_active_counters (error_code &ec = throws)
```

Resets all active performance counters.

**Note** As long as `ec` is not pre-initialized to `hpx::throws` this function doesn’t throw but returns the result code using the parameter `ec`. Otherwise it throws an instance of `hpx::exception`.

**Note** The active counters are those which have been specified on the command line while executing the application (see command line option –hpx:print-counter)

**Parameters**

- **ec**: [in,out] this represents the error status on exit, if this is pre-initialized to `hpx::throws` the function will throw on error instead.

```cpp
void reinit_active_counters (bool reset = true, error_code &ec = throws)
```

Re-initialize all active performance counters.

**Note** As long as `ec` is not pre-initialized to `hpx::throws` this function doesn’t throw but returns the result code using the parameter `ec`. Otherwise it throws an instance of `hpx::exception`.

**Note** The active counters are those which have been specified on the command line while executing the application (see command line option –hpx:print-counter)

**Parameters**

- **reset**: [in] Reset the current values before re-initializing counters (default: true)
- **ec**: [in,out] this represents the error status on exit, if this is pre-initialized to `hpx::throws` the function will throw on error instead.

```cpp
void stop_active_counters (error_code &ec = throws)
```

Stop all active performance counters.

**Note** As long as `ec` is not pre-initialized to `hpx::throws` this function doesn’t throw but returns the result code using the parameter `ec`. Otherwise it throws an instance of `hpx::exception`.

**Note** The active counters are those which have been specified on the command line while executing the application (see command line option –hpx:print-counter)

**Parameters**

- **ec**: [in,out] this represents the error status on exit, if this is pre-initialized to `hpx::throws` the function will throw on error instead.

```cpp
void evaluate_active_counters (bool reset = false, char const *description = nullptr, error_code &ec = throws)
```

Evaluate and output all active performance counters, optionally naming the point in code marked by this function.

**Note** As long as `ec` is not pre-initialized to `hpx::throws` this function doesn’t throw but returns the result code using the parameter `ec`. Otherwise it throws an instance of `hpx::exception`.

2.9. API reference 285
**Note** The output generated by this function is redirected to the destination specified by the corresponding command line options (see `--hpx:print-counter-destination`).

**Note** The active counters are those which have been specified on the command line while executing the application (see command line option `--hpx:print-counter`)

**Parameters**

- **reset**: [in] this is an optional flag allowing to reset the counter value after it has been evaluated.
- **description**: [in] this is an optional value naming the point in the code marked by the call to this function.
- **ec**: [in,out] this represents the error status on exit, if this is pre-initialized to `hpx::throws` the function will throw on error instead.

```cpp
serialization::binary_filter *create_binary_filter (char const *binary_filter_type, bool compress, serialization::binary_filter *next_filter = nullptr, error_code &ec = throws)
```

Create an instance of a binary filter plugin.

**Note** As long as `ec` is not pre-initialized to `hpx::throws` this function doesn’t throw but returns the result code using the parameter `ec`. Otherwise it throws an instance of `hpx::exception`.

**Parameters**

- **binary_filter_type**: [in] The type of the binary filter to create
- **compress**: [in] The created filter should support compression
- **next_filter**: [in] Use this as the filter to dispatch the invocation into.
- **ec**: [in,out] this represents the error status on exit, if this is pre-initialized to `hpx::throws` the function will throw on error instead.

```cpp
std::vector<Client> find_all_from_basename (std::string base_name, std::size_t num_ids)
```

Return all registered clients from all localities from the given base name.

This function locates all ids which were registered with the given base name. It returns a list of futures representing those ids.

Return all registered clients from all localities from the given base name.

**Return** A list of futures representing the ids which were registered using the given base name.

**Note** The futures will become ready even if the event (for instance, binding the name to an id) has already happened in the past. This is important in order to reliably retrieve ids from a name, even if the name was already registered.

**Parameters**

- **base_name**: [in] The base name for which to retrieve the registered ids.
- **num_ids**: [in] The number of registered ids to expect.

This function locates all ids which were registered with the given base name. It returns a list of futures representing those ids.

**Return** A list of futures representing the ids which were registered using the given base name.

**Note** The futures embedded in the returned client objects will become ready even if the event (for instance, binding the name to an id) has already happened in the past. This is important in order to reliably retrieve ids from a name, even if the name was already registered.
Template Parameters

- Client: The client type to return

Parameters

- base_name: [in] The base name for which to retrieve the registered ids.
- num_ids: [in] The number of registered ids to expect.

`std::vector<Client> find_from_basename (std::string base_name, std::vector< std::size_t > const & ids)`

Return registered ids from the given base name and sequence numbers.

This function locates the ids which were registered with the given base name and the given sequence numbers. It returns a list of futures representing those ids.

Return A list of futures representing the ids which were registered using the given base name and sequence numbers.

Note The futures will become ready even if the event (for instance, binding the name to an id) has already happened in the past. This is important in order to reliably retrieve ids from a name, even if the name was already registered.

Parameters

- base_name: [in] The base name for which to retrieve the registered ids.
- ids: [in] The sequence numbers of the registered ids.

This function locates the ids which were registered with the given base name and the given sequence numbers. It returns a list of futures representing those ids.

Return A list of futures representing the ids which were registered using the given base name and sequence numbers.

Note The futures embedded in the returned client objects will become ready even if the event (for instance, binding the name to an id) has already happened in the past. This is important in order to reliably retrieve ids from a name, even if the name was already registered.

Template Parameters

- Client: The client type to return

Parameters

- base_name: [in] The base name for which to retrieve the registered ids.
- ids: [in] The sequence numbers of the registered ids.

Client `find_from_basename (std::string base_name, std::size_t sequence_nr = ~static_cast<std::size_t>(0))`

Return registered id from the given base name and sequence number.

This function locates the id which was registered with the given base name and the given sequence number. It returns a future representing those id.

This function locates the id which was registered with the given base name and the given sequence number. It returns a future representing those id.

Return A representing the id which was registered using the given base name and sequence numbers.
**Note** The future will become ready even if the event (for instance, binding the name to an id) has already happened in the past. This is important in order to reliably retrieve ids from a name, even if the name was already registered.

**Parameters**
- **base_name**: [in] The base name for which to retrieve the registered ids.
- **sequence_nr**: [in] The sequence number of the registered id.

**Return** A representing the id which was registered using the given base name and sequence numbers.

**Note** The future embedded in the returned client object will become ready even if the event (for instance, binding the name to an id) has already happened in the past. This is important in order to reliably retrieve ids from a name, even if the name was already registered.

**Template Parameters**
- **Client**: The client type to return

**Parameters**
- **base_name**: [in] The base name for which to retrieve the registered ids.
- **sequence_nr**: [in] The sequence number of the registered id.

```cpp
hpx::future<bool> register_with_basename (std::string base_name, hpx::id_type id, std::size_t sequence_nr = ~static_cast<std::size_t>(0))
```

Register the given id using the given base name.

The function registers the given ids using the provided base name.

**Return** A future representing the result of the registration operation itself.

**Note** The operation will fail if the given sequence number is not unique.

**Parameters**
- **base_name**: [in] The base name for which to retrieve the registered ids.
- **id**: [in] The id to register using the given base name.
- **sequence_nr**: [in, optional] The sequential number to use for the registration of the id. This number has to be unique system wide for each registration using the same base name. The default is the current locality identifier. Also, the sequence numbers have to be consecutive starting from zero.

```cpp
hpx::future<bool> register_with_basename (std::string base_name, hpx::future<hpx::id_type> f, std::size_t sequence_nr = ~static_cast<std::size_t>(0))
```

Register the id wrapped in the given future using the given base name.

The function registers the object the given future refers to using the provided base name.

**Return** A future representing the result of the registration operation itself.

**Note** The operation will fail if the given sequence number is not unique.

**Parameters**
- **base_name**: [in] The base name for which to retrieve the registered ids.
- **f**: [in] The future which should be registered using the given base name.
• sequence_nr: [in, optional] The sequential number to use for the registration of the id. This number has to be unique system wide for each registration using the same base name. The default is the current locality identifier. Also, the sequence numbers have to be consecutive starting from zero.

`template <typename Client, typename Stub> hpx::future<bool> register_with_basename(std::string base_name, components::client_base<Client, Stub> &client, std::size_t sequence_nr = ~static_cast<std::size_t>(0))`

Register the id wrapped in the given client using the given base name.

The function registers the object the given client refers to using the provided base name.

**Return** A future representing the result of the registration operation itself.

**Note** The operation will fail if the given sequence number is not unique.

**Template Parameters**
- Client: The client type to register

**Parameters**
- base_name: [in] The base name for which to retrieve the registered ids.
- client: [in] The client which should be registered using the given base name.
- sequence_nr: [in, optional] The sequential number to use for the registration of the id. This number has to be unique system wide for each registration using the same base name. The default is the current locality identifier. Also, the sequence numbers have to be consecutive starting from zero.

Client `unregister_with_basename(std::string base_name, std::size_t sequence_nr = ~static_cast<std::size_t>(0))`

Unregister the given id using the given base name.

The function unregisters the given ids using the provided base name.

**Return** A future representing the result of the un-registration operation itself.

**Parameters**
- base_name: [in] The base name for which to retrieve the registered ids.
- sequence_nr: [in, optional] The sequential number to use for the un-registration. This number has to be the same as has been used with `register_with_basename` before.

The function unregisters the given ids using the provided base name.

**Return** A future representing the result of the un-registration operation itself.

**Template Parameters**
- Client: The client type to return

**Parameters**
- base_name: [in] The base name for which to retrieve the registered ids.
- sequence_nr: [in, optional] The sequential number to use for the un-registration. This number has to be the same as has been used with `register_with_basename` before.
naming::id_type find_here(error_code &ec = throws)

Return the global id representing this locality.

The function find_here() can be used to retrieve the global id usable to refer to the current locality.

Note Generally, the id of a locality can be used for instance to create new instances of components and to invoke plain actions (global functions).

Return The global id representing the locality this function has been called on.

Note As long as ec is not pre-initialized to hpx::throws this function doesn’t throw but returns the result code using the parameter ec. Otherwise it throws an instance of hpx::exception.

Note This function will return meaningful results only if called from an HPX-thread. It will return hpx::naming::invalid_id otherwise.

See hpx::find_all_localities(), hpx::find_locality()

Parameters

• ec: [in,out] this represents the error status on exit, if this is pre-initialized to hpx::throws the function will throw on error instead.

naming::id_type find_root_locality(error_code &ec = throws)

Return the global id representing the root locality.

The function find_root_locality() can be used to retrieve the global id usable to refer to the root locality. The root locality is the locality where the main AGAS service is hosted.

Note Generally, the id of a locality can be used for instance to create new instances of components and to invoke plain actions (global functions).

Return The global id representing the root locality for this application.

Note As long as ec is not pre-initialized to hpx::throws this function doesn’t throw but returns the result code using the parameter ec. Otherwise it throws an instance of hpx::exception.

Note This function will return meaningful results only if called from an HPX-thread. It will return hpx::naming::invalid_id otherwise.

See hpx::find_all_localities(), hpx::find_locality()

Parameters

• ec: [in,out] this represents the error status on exit, if this is pre-initialized to hpx::throws the function will throw on error instead.

std::vector<naming::id_type> find_all_localities(error_code &ec = throws)

Return the list of global ids representing all localities available to this application.

The function find_all_localities() can be used to retrieve the global ids of all localities currently available to this application.

Note Generally, the id of a locality can be used for instance to create new instances of components and to invoke plain actions (global functions).

Return The global ids representing the localities currently available to this application.

Note As long as ec is not pre-initialized to hpx::throws this function doesn’t throw but returns the result code using the parameter ec. Otherwise it throws an instance of hpx::exception.
Note This function will return meaningful results only if called from an HPX-thread. It will return an empty vector otherwise.

See hpx::find_here(), hpx::find_locality()

Parameters

- **ec**: [in,out] this represents the error status on exit, if this is pre-initialized to hpx::throws the function will throw on error instead.

```c++
std::vector<naming::id_type> find_all_localities(components::component_type type, error_code &ec = throws)
```

Return the list of global ids representing all localities available to this application which support the given component type.

The function find_all_localities() can be used to retrieve the global ids of all localities currently available to this application which support the creation of instances of the given component type.

Note Generally, the id of a locality can be used for instance to create new instances of components and to invoke plain actions (global functions).

Return The global ids representing the localities currently available to this application which support the creation of instances of the given component type. If no localities supporting the given component type are currently available, this function will return an empty vector.

Note As long as ec is not pre-initialized to hpx::throws this function doesn’t throw but returns the result code using the parameter ec. Otherwise it throws an instance of hpx::exception.

Note This function will return meaningful results only if called from an HPX-thread. It will return an empty vector otherwise.

See hpx::find_here(), hpx::find_locality()

Parameters

- **type**: [in] The type of the components for which the function should return the available localities.

```c++
std::vector<naming::id_type> find_remote_localities(error_code &ec = throws)
```

Return the list of locality ids of remote localities supporting the given component type. By default this function will return the list of all remote localities (all but the current locality).

The function find_remote_localities() can be used to retrieve the global ids of all remote localities currently available to this application (i.e. all localities except the current one).

Note Generally, the id of a locality can be used for instance to create new instances of components and to invoke plain actions (global functions).

Return The global ids representing the remote localities currently available to this application.

Note As long as ec is not pre-initialized to hpx::throws this function doesn’t throw but returns the result code using the parameter ec. Otherwise it throws an instance of hpx::exception.

Note This function will return meaningful results only if called from an HPX-thread. It will return an empty vector otherwise.

See hpx::find_here(), hpx::find_locality()
• ec: [in,out] this represents the error status on exit, if this is pre-initialized to hpx::throws the function will throw on error instead.

`std::vector<naming::id_type> find_remote_localities (components::component_type type, error_code &ec = throws)`

Return the list of locality ids of remote localities supporting the given component type. By default this function will return the list of all remote localities (all but the current locality).

The function find_remote_localities() can be used to retrieve the global ids of all remote localities currently available to this application (i.e. all localities except the current one) which support the creation of instances of the given component type.

**Note** Generally, the id of a locality can be used for instance to create new instances of components and to invoke plain actions (global functions).

**Return** The global ids representing the remote localities currently available to this application.

**Note** As long as ec is not pre-initialized to hpx::throws this function doesn’t throw but returns the result code using the parameter ec. Otherwise it throws an instance of hpx::exception.

**Note** This function will return meaningful results only if called from an HPX-thread. It will return an empty vector otherwise.

**See** hpx::find_here(), hpx::find_locality()

**Parameters**

- **type**: [in] The type of the components for which the function should return the available remote localities.
- **ec**: [in,out] this represents the error status on exit, if this is pre-initialized to hpx::throws the function will throw on error instead.

`naming::id_type find_locality (components::component_type type, error_code &ec = throws)`

Return the global id representing an arbitrary locality which supports the given component type.

The function find_locality() can be used to retrieve the global id of an arbitrary locality currently available to this application which supports the creation of instances of the given component type.

**Note** Generally, the id of a locality can be used for instance to create new instances of components and to invoke plain actions (global functions).

**Return** The global id representing an arbitrary locality currently available to this application which supports the creation of instances of the given component type. If no locality supporting the given component type is currently available, this function will return hpx::naming::invalid_id.

**Note** As long as ec is not pre-initialized to hpx::throws this function doesn’t throw but returns the result code using the parameter ec. Otherwise it throws an instance of hpx::exception.

**Note** This function will return meaningful results only if called from an HPX-thread. It will return hpx::naming::invalid_id otherwise.

**See** hpx::find_here(), hpx::find_all_localities()

**Parameters**

- **type**: [in] The type of the components for which the function should return any available locality.
- **ec**: [in,out] this represents the error status on exit, if this is pre-initialized to hpx::throws the function will throw on error instead.
naming::id_type get_colocation_id(launch::sync_policy, naming::id_type const &id, error_code &ec = throws)

Return the id of the locality where the object referenced by the given id is currently located.

The function hpx::get_colocation_id() returns the id of the locality where the given object is currently located.

Note As long as ec is not pre-initialized to hpx::throws this function doesn’t throw but returns the result code using the parameter ec. Otherwise it throws an instance of hpx::exception.

See hpx::get_colocation_id()

Parameters

- id: [in] The id of the object to locate.
- ec: [in,out] this represents the error status on exit, if this is pre-initialized to hpx::throws the function will throw on error instead.

lcos::future<naming::id_type> get_colocation_id(naming::id_type const &id)

Asynchronously return the id of the locality where the object referenced by the given id is currently located on.

See hpx::get_colocation_id(launch::sync_policy)

Parameters

- id: [in] The id of the object to locate.

**template <typename Component>**

hpx::future<std::shared_ptr<Component>> get_ptr(naming::id_type const &id)

Returns a future referring to the pointer to the underlying memory of a component.

The function hpx::get_ptr can be used to extract a future referring to the pointer to the underlying memory of a given component.

Return This function returns a future representing the pointer to the underlying memory for the component instance with the given id.

Note This function will successfully return the requested result only if the given component is currently located on the calling locality. Otherwise the function will raise an error.

Note The component instance the returned pointer refers to can not be migrated as long as there is at least one copy of the returned shared_ptr alive.

Parameters

- id: [in] The global id of the component for which the pointer to the underlying memory should be retrieved.

Template Parameters

- The: only template parameter has to be the type of the server side component.

**template <typename Derived, typename Stub>**

hpx::future<std::shared_ptr<typename components::client_base<Derived, Stub>::server_component_type>> get_ptr(components::client_base<Derived, Stub> const &c)

Returns a future referring to the pointer to the underlying memory of a component.
The function `hpx::get_ptr` can be used to extract a future referring to the pointer to the underlying memory of a given component.

**Return** This function returns a future representing the pointer to the underlying memory for the component instance with the given `id`.

**Note** This function will successfully return the requested result only if the given component is currently located on the calling locality. Otherwise the function will raise an error.

**Note** The component instance the returned pointer refers to can not be migrated as long as there is at least one copy of the returned `shared_ptr` alive.

**Parameters**
- `c`: [in] A client side representation of the component for which the pointer to the underlying memory should be retrieved.

```template <typename Component>
std::shared_ptr<Component> get_ptr (launch::sync_policy p, naming::id_type const &id, error_code &ec = throws)
```

Returns the pointer to the underlying memory of a component.

The function `hpx::get_ptr_sync` can be used to extract the pointer to the underlying memory of a given component.

**Return** This function returns the pointer to the underlying memory for the component instance with the given `id`.

**Note** This function will successfully return the requested result only if the given component is currently located on the requesting locality. Otherwise the function will raise and error.

**Note** The component instance the returned pointer refers to can not be migrated as long as there is at least one copy of the returned `shared_ptr` alive.

**Note** As long as `ec` is not pre-initialized to `hpx::throws` this function doesn’t throw but returns the result code using the parameter `ec`. Otherwise it throws an instance of `hpx::exception`.

**Parameters**
- `p`: [in] The parameter `p` represents a placeholder type to turn make the call synchronous.
- `id`: [in] The global id of the component for which the pointer to the underlying memory should be retrieved.
- `ec`: [in,out] this represents the error status on exit, if this is pre-initialized to `hpx::throws` the function will throw on error instead.

**Template Parameters**
- The: only template parameter has to be the type of the server side component.

```template <typename Derived, typename Stub>`
Returns the pointer to the underlying memory of a component.

The function hpx::get_ptr_sync can be used to extract the pointer to the underlying memory of a given component.

**Return** This function returns the pointer to the underlying memory for the component instance with the given id.

**Note** This function will successfully return the requested result only if the given component is currently located on the requesting locality. Otherwise the function will raise and error.

**Note** The component instance the returned pointer refers to can not be migrated as long as there is at least one copy of the returned shared_ptr alive.

**Note** As long as ec is not pre-initialized to hpx::throws this function doesn’t throw but returns the result code using the parameter ec. Otherwise it throws an instance of hpx::exception.

**Parameters**
- p: [in] The parameter p represents a placeholder type to turn make the call synchronous.
- c: [in] A client side representation of the component for which the pointer to the underlying memory should be retrieved.
- ec: [in,out] this represents the error status on exit, if this is pre-initialized to hpx::throws the function will throw on error instead.

---

```
std::uint32_t get_locality_id(error_code &ec = throws)
```

Return the number of the locality this function is being called from.

This function returns the id of the current locality.

**Note** The returned value is zero based and its maximum value is smaller than the overall number of localities the current application is running on (as returned by get_num_localities()).

**Note** As long as ec is not pre-initialized to hpx::throws this function doesn’t throw but returns the result code using the parameter ec. Otherwise it throws an instance of hpx::exception.

**Note** This function needs to be executed on a HPX-thread. It will fail otherwise (it will return -1).

**Parameters**
- ec: [in,out] this represents the error status on exit, if this is pre-initialized to hpx::throws the function will throw on error instead.

---

```
std::string get_locality_name()
```

Return the name of the locality this function is called on.
This function returns the name for the locality on which this function is called.

**Return** This function returns the name for the locality on which the function is called. The name is retrieved from the underlying networking layer and may be different for different parcelports.

**See** `future<std::string> get_locality_name(naming::id_type const &id)`

`future<std::string> get_locality_name(naming::id_type const &id)`

Return the name of the referenced locality.

This function returns a future referring to the name for the locality of the given id.

**Return** This function returns the name for the locality of the given id. The name is retrieved from the underlying networking layer and may be different for different parcel ports.

**See** `std::string get_locality_name()`

**Parameters**

- `id`: [in] The global id of the locality for which the name should be retrieved

`std::uint32_t get_initial_num_localities()`

Return the number of localities which were registered at startup for the running application.

The function `get_initial_num_localities` returns the number of localities which were connected to the console at application startup.

**Note** As long as `ec` is not pre-initialized to `hpx::throws` this function doesn’t throw but returns the result code using the parameter `ec`. Otherwise it throws an instance of `hpx::exception`.

**See** `hpx::find_all_localities`, `hpx::get_num_localities`

`lcos::future<std::uint32_t> get_num_localities()`

Asynchronously return the number of localities which are currently registered for the running application.

The function `get_num_localities` asynchronously returns the number of localities currently connected to the console. The returned future represents the actual result.

**Note** This function will return meaningful results only if called from an HPX-thread. It will return 0 otherwise.

**See** `hpx::find_all_localities`, `hpx::get_num_localities`

`std::uint32_t get_num_localities(launch::sync_policy, error_code &ec = throws)`

Return the number of localities which are currently registered for the running application.

The function `get_num_localities` returns the number of localities currently connected to the console.

**Note** This function will return meaningful results only if called from an HPX-thread. It will return 0 otherwise.

**Note** As long as `ec` is not pre-initialized to `hpx::throws` this function doesn’t throw but returns the result code using the parameter `ec`. Otherwise it throws an instance of `hpx::exception`.

**See** `hpx::find_all_localities`, `hpx::get_num_localities`

**Parameters**

- `ec`: [in,out] this represents the error status on exit, if this is pre-initialized to `hpx::throws` the function will throw on error instead.
`lcos::future<std::uint32_t> get_num_localities (components::component_type t)`

Asynchronously return the number of localities which are currently registered for the running application.

The function `get_num_localities` asynchronously returns the number of localities currently connected to the console which support the creation of the given component type. The returned future represents the actual result.

**Note** This function will return meaningful results only if called from an HPX-thread. It will return 0 otherwise.

**See** `hpx::find_all_localities, hpx::get_num_localities`

**Parameters**

• `t`: The component type for which the number of connected localities should be retrieved.

`std::uint32_t get_num_localities (launch::sync_policy, components::component_type t, error_code &ec = throws)`

Synchronously return the number of localities which are currently registered for the running application.

The function `get_num_localities` returns the number of localities currently connected to the console which support the creation of the given component type. The returned future represents the actual result.

**Note** This function will return meaningful results only if called from an HPX-thread. It will return 0 otherwise.

**See** `hpx::find_all_localities, hpx::get_num_localities`

**Parameters**

• `t`: The component type for which the number of connected localities should be retrieved.

• `ec`: [in,out] this represents the error status on exit, if this is pre-initialized to `hpx::throws` the function will throw on error instead.

`std::size_t get_os_thread_count ()`

Return the number of OS-threads running in the runtime instance the current HPX-thread is associated with.

`std::size_t get_os_thread_count (threads::executor const &exec)`

Return the number of worker OS-threads used by the given executor to execute HPX threads.

This function returns the number of cores used to execute HPX threads for the given executor. If the function is called while no HPX runtime system is active, it will return zero. If the executor is not valid, this function will fall back to retrieving the number of OS threads used by HPX.

**Parameters**

• `exec`: [in] The executor to be used.

`std::string get_thread_name ()`

Return the name of the calling thread.

This function returns the name of the calling thread. This name uniquely identifies the thread in the context of HPX. If the function is called while no HPX runtime system is active, the result will be "<unknown>".

`std::size_t get_worker_thread_num ()`

Return the number of the current OS-thread running in the runtime instance the current HPX-thread is executed with.

This function returns the zero based index of the OS-thread which executes the current HPX-thread.
Note The returned value is zero based and its maximum value is smaller than the overall number of OS-
threads executed (as returned by get_os_thread_count()).

Note This function needs to be executed on a HPX-thread. It will fail otherwise (it will return -1).

```cpp
std::size_t get_worker_thread_num (error_code &ec)
```
Return the number of the current OS-thread running in the runtime instance the current HPX-thread is
executed with.

This function returns the zero based index of the OS-thread which executes the current HPX-thread.

Note The returned value is zero based and its maximum value is smaller than the overall number of OS-
threads executed (as returned by get_os_thread_count()). It will return -1 if the current thread is not a
known thread or if the runtime is not in running state.

Note This function needs to be executed on a HPX-thread. It will fail otherwise (it will return -1).

**Parameters**

- `ec`: [in,out] this represents the error status on exit.

```cpp
void report_error (std::size_t num_thread, std::exception_ptr const &e)
```
The function report_error reports the given exception to the console.

```cpp
void report_error (std::exception_ptr const &e)
```
The function report_error reports the given exception to the console.

```cpp
char const *get_runtime_mode_name (runtime_mode state)
```
Get the readable string representing the name of the given runtime_mode constant.

```cpp
runtime_mode get_runtime_mode_from_name (std::string const &mode)
```
Returns the internal representation (runtime_mode constant) from the readable string representing the
name.

This represents the internal representation from the readable string representing the name.

**Parameters**

- `mode`: this represents the runtime mode

```cpp
void register_pre_shutdown_function (shutdown_function_type f)
```
Add a function to be executed by a HPX thread during hpx::finalize() but guaranteed before any shutdown
function is executed (system-wide)

Any of the functions registered with register_pre_shutdown_function are guaranteed to be executed by an
HPX thread during the execution of hpx::finalize() before any of the registered shutdown functions are
executed (see: hpx::register_shutdown_function()).

**Note** If this function is called while the pre-shutdown functions are being executed, or after that point, it
will raise a invalid_status exception.

**See** hpx::register_shutdown_function()

**Parameters**

- `f`: [in] The function to be registered to run by an HPX thread as a pre-shutdown function.
void `register_shutdown_function`(`shutdown_function_type f`)
Add a function to be executed by a HPX thread during `hpx::finalize()` but guaranteed after any pre-shutdown function is executed (system-wide).

Any of the functions registered with `register_shutdown_function` are guaranteed to be executed by an HPX thread during the execution of `hpx::finalize()` after any of the registered pre-shutdown functions are executed (see: `hpx::register_pre_shutdown_function()`).

**Note** If this function is called while the shutdown functions are being executed, or after that point, it will raise a `invalid_status` exception.

**See** `hpx::register_pre_shutdown_function()`

**Parameters**
- `f`: [in] The function to be registered to run by an HPX thread as a shutdown function.

void `register_pre_startup_function`(`startup_function_type f`)
Add a function to be executed by a HPX thread before `hpx_main` but guaranteed before any startup function is executed (system-wide).

Any of the functions registered with `register_pre_startup_function` are guaranteed to be executed by an HPX thread before any of the registered startup functions are executed (see `hpx::register_startup_function()`).

This function is one of the few API functions which can be called before the runtime system has been fully initialized. It will automatically stage the provided startup function to the runtime system during its initialization (if necessary).

**Note** If this function is called while the pre-startup functions are being executed or after that point, it will raise a `invalid_status` exception.

**Parameters**
- `f`: [in] The function to be registered to run by an HPX thread as a pre-startup function.

**See** `hpx::register_startup_function()`

void `register_startup_function`(`startup_function_type f`)
Add a function to be executed by a HPX thread before `hpx_main` but guaranteed after any pre-startup function is executed (system-wide).

Any of the functions registered with `register_startup_function` are guaranteed to be executed by an HPX thread after any of the registered pre-startup functions are executed (see: `hpx::register_pre_startup_function()`), but before `hpx_main` is being called.

This function is one of the few API functions which can be called before the runtime system has been fully initialized. It will automatically stage the provided startup function to the runtime system during its initialization (if necessary).

**Note** If this function is called while the startup functions are being executed or after that point, it will raise a `invalid_status` exception.

**Parameters**
- `f`: [in] The function to be registered to run by an HPX thread as a startup function.

**See** `hpx::register_pre_startup_function()`
void trigger_lco_event(naming::id_type const &id, naming::address &addr, bool move_credits = true)

Trigger the LCO referenced by the given id.

Parameters

• id: [in] This represents the id of the LCO which should be triggered.
• addr: [in] This represents the addr of the LCO which should be triggered.
• move_credits: [in] If this is set to true then it is ok to send all credits in id along with the generated message. The default value is true.

void trigger_lco_event(naming::id_type const &id, bool move_credits = true)

Trigger the LCO referenced by the given id.

Parameters

• id: [in] This represents the id of the LCO which should be triggered.
• move_credits: [in] If this is set to true then it is ok to send all credits in id along with the generated message. The default value is true.

void trigger_lco_event(naming::id_type const &id, naming::address &addr, naming::id_type const &cont, bool move_credits = true)

Trigger the LCO referenced by the given id.

Parameters

• id: [in] This represents the id of the LCO which should be triggered.
• addr: [in] This represents the addr of the LCO which should be triggered.
• cont: [in] This represents the LCO to trigger after completion.
• move_credits: [in] If this is set to true then it is ok to send all credits in id along with the generated message. The default value is true.

void trigger_lco_event(naming::id_type const &id, naming::id_type const &cont, bool move_credits = true)

Trigger the LCO referenced by the given id.

Parameters

• id: [in] This represents the id of the LCO which should be triggered.
• cont: [in] This represents the LCO to trigger after completion.
• move_credits: [in] If this is set to true then it is ok to send all credits in id along with the generated message. The default value is true.

template<typename Result>
void set_lco_value(naming::id_type const &id, naming::address &addr, Result &&t, bool move_credits = true)

Set the result value for the LCO referenced by the given id.

Parameters

• id: [in] This represents the id of the LCO which should receive the given value.
• addr: [in] This represents the addr of the LCO which should be triggered.
• \( t \): \[ \text{in} \] This is the value which should be sent to the LCO.

• \text{move\_credits}: \[ \text{in} \] If this is set to \text{true} then it is ok to send all credits in \( id \) along with the generated message. The default value is \text{true}.

\[
\text{template } <\text{typename Result}> \\
\text{std::enable_if<!std::is_same<\text{typename util::decay<Result>::type, naming::address>::value>::type set_lco_value(naming::id_type const &id, Result &&t, bool move_credits = true)}
\]

Set the result value for the (managed) LCO referenced by the given id.

\textbf{Parameters}

• \( id \): \[ \text{in} \] This represents the id of the LCO which should receive the given value.

• \( t \): \[ \text{in} \] This is the value which should be sent to the LCO.

• \text{move\_credits}: \[ \text{in} \] If this is set to \text{true} then it is ok to send all credits in \( id \) along with the generated message. The default value is \text{true}.

\[
\text{template } <\text{typename Result}> \\
\text{std::enable_if<!std::is_same<\text{typename util::decay<Result>::type, naming::address>::value>::type set_lco_value_unmanaged(naming::id_type const &id, Result &&t, bool move_credits = true)}
\]

Set the result value for the (unmanaged) LCO referenced by the given id.

\textbf{Parameters}

• \( id \): \[ \text{in} \] This represents the id of the LCO which should receive the given value.

• \( addr \): \[ \text{in} \] This represents the addr of the LCO which should be triggered.

\[
\text{template } <\text{typename Result}> \\
\text{void set_lco_value(naming::id_type const &id, naming::address &addr, Result &&t, naming::id_type const &cont, bool move_credits = true)}
\]

Set the result value for the LCO referenced by the given id.

\textbf{Parameters}

• \( id \): \[ \text{in} \] This represents the id of the LCO which should receive the given value.

• \( addr \): \[ \text{in} \] This represents the addr of the LCO which should be triggered.
• \texttt{t}: \texttt{[in]} This is the value which should be sent to the LCO.
• \texttt{cont}: \texttt{[in]} This represents the LCO to trigger after completion.
• \texttt{move\_credits}: \texttt{[in]} If this is set to \texttt{true} then it is ok to send all credits in \texttt{id} along with the generated message. The default value is \texttt{true}.

\texttt{template <typename Result> std::enable_if<!std::is_same<typename util::decay<Result>::type, naming::address>::value>::type set\_lco\_value(naming::id\_type const& \texttt{id}, Result&& \texttt{t}, naming::id\_type const& \texttt{cont}, bool move\_credits = true)}

Set the result value for the (managed) LCO referenced by the given id.

Parameters

• \texttt{id}: \texttt{[in]} This represents the id of the LCO which should receive the given value.
• \texttt{t}: \texttt{[in]} This is the value which should be sent to the LCO.
• \texttt{cont}: \texttt{[in]} This represents the LCO to trigger after completion.
• \texttt{move\_credits}: \texttt{[in]} If this is set to \texttt{true} then it is ok to send all credits in \texttt{id} along with the generated message. The default value is \texttt{true}.

\texttt{template <typename Result> std::enable_if<!std::is_same<typename util::decay<Result>::type, naming::address>::value>::type set\_lco\_value\_unmanaged(naming::id\_type const& \texttt{id}, Result&& \texttt{t}, naming::id\_type const& \texttt{cont}, bool move\_credits = true)}

Set the result value for the (unmanaged) LCO referenced by the given id.

Parameters

• \texttt{id}: \texttt{[in]} This represents the id of the LCO which should receive the given value.
• \texttt{t}: \texttt{[in]} This is the value which should be sent to the LCO.
• \texttt{cont}: \texttt{[in]} This represents the LCO to trigger after completion.
void set_lco_error(naming::id_type const &id, naming::address &addr, std::exception_ptr const &e, bool move_credits = true)
Set the error state for the LCO referenced by the given id.

Parameters

• id: [in] This represents the id of the LCO which should receive the error value.
• addr: [in] This represents the addr of the LCO which should be triggered.
• e: [in] This is the error value which should be sent to the LCO.
• move_credits: [in] If this is set to true then it is ok to send all credits in id along with the generated message. The default value is true.
void set_lco_error(naming::id_type const &id, naming::address &&addr, std::exception_ptr const &e, naming::id_type const &cont, bool move_credits = true)

Set the error state for the LCO referenced by the given id.

Parameters

- id: [in] This represents the id of the LCO which should receive the error value.
- addr: [in] This represents the addr of the LCO which should be triggered.
- e: [in] This is the error value which should be sent to the LCO.
- cont: [in] This represents the LCO to trigger after completion.
- move_credits: [in] If this is set to true then it is ok to send all credits in id along with the generated message. The default value is true.

void set_lco_error(naming::id_type const &id, naming::address &&addr, std::exception_ptr &&e, naming::id_type const &cont, bool move_credits = true)

Set the error state for the LCO referenced by the given id.

Parameters

- id: [in] This represents the id of the LCO which should receive the error value.
- addr: [in] This represents the addr of the LCO which should be triggered.
- e: [in] This is the error value which should be sent to the LCO.
- cont: [in] This represents the LCO to trigger after completion.
- move_credits: [in] If this is set to true then it is ok to send all credits in id along with the generated message. The default value is true.

void set_lco_error(naming::id_type const &id, std::exception_ptr const &e, naming::id_type const &cont, bool move_credits = true)

Set the error state for the LCO referenced by the given id.

Parameters

- id: [in] This represents the id of the LCO which should receive the error value.
- e: [in] This is the error value which should be sent to the LCO.
- cont: [in] This represents the LCO to trigger after completion.
- move_credits: [in] If this is set to true then it is ok to send all credits in id along with the generated message. The default value is true.

void set_lco_error(naming::id_type const &id, std::exception_ptr &&e, naming::id_type const &cont, bool move_credits = true)

Set the error state for the LCO referenced by the given id.

Parameters

- id: [in] This represents the id of the LCO which should receive the error value.
- e: [in] This is the error value which should be sent to the LCO.
- cont: [in] This represents the LCO to trigger after completion.
- move_credits: [in] If this is set to true then it is ok to send all credits in id along with the generated message. The default value is true.
template <typename Component, typename ... Ts>
<unspecified> hpx::new_(id_type const & locality, Ts &... vs)
Create one or more new instances of the given Component type on the specified locality.

This function creates one or more new instances of the given Component type on the specified locality and returns a future object for the global address which can be used to reference the new component instance.

Note This function requires to specify an explicit template argument which will define what type of component(s) to create, for instance:

```cpp
hpx::future<hpx::id_type> f =
    hpx::new_<some_component>(hpx::find_here(), ...);
    hpx::id_type id = f.get();
```

Return The function returns different types depending on its use:

- If the explicit template argument Component represents a component type (traits::is_component<Component>::value evaluates to true), the function will return an hpx::future object instance which can be used to retrieve the global address of the newly created component.
- If the explicit template argument Component represents a client side object (traits::is_client<Component>::value evaluates to true), the function will return a new instance of that type which can be used to refer to the newly created component instance.

Parameters

- locality: [in] The global address of the locality where the new instance should be created on.
- vs: [in] Any number of arbitrary arguments (passed by value, by const reference or by rvalue reference) which will be forwarded to the constructor of the created component instance.

---

template <typename Component, typename ... Ts>
<unspecified> hpx::local_new(Ts &... vs)
Create one new instance of the given Component type on the current locality.

This function creates one new instance of the given Component type on the current locality and returns a future object for the global address which can be used to reference the new component instance.

Note This function requires to specify an explicit template argument which will define what type of component(s) to create, for instance:

```cpp
hpx::future<hpx::id_type> f =
    hpx::local_new<some_component>(...);
    hpx::id_type id = f.get();
```

Return The function returns different types depending on its use:

- If the explicit template argument Component represents a component type (traits::is_component<Component>::value evaluates to true), the function will return an hpx::future object instance which can be used to retrieve the global address of the newly created component. If the first argument is hpx::launch::sync the function will directly return an hpx::id_type.
- If the explicit template argument Component represents a client side object (traits::is_client<Component>::value evaluates to true), the function will return a new instance of that type which can be used to refer to the newly created component instance.
**Note** The difference of this function to `hpx::new_` is that it can be used in cases where the supplied arguments are non-copyable and non-movable. All operations are guaranteed to be local only.

**Parameters**
- `vs`: [in] Any number of arbitrary arguments (passed by value, by const reference or by rvalue reference) which will be forwarded to the constructor of the created component instance.

```cpp
template <typename Component, typename typename ... Ts>
<unspecified> hpx::new_(id_type const & locality, std::size_t count, Ts &&... vs)
```

Create multiple new instances of the given Component type on the specified locality.

This function creates multiple new instances of the given Component type on the specified locality and returns a future object for the global address which can be used to reference the new component instance.

**Note** This function requires to specify an explicit template argument which will define what type of component(s) to create, for instance:

```cpp
hpx::future<std::vector<hpx::id_type> > f =
  hpx::new_<some_component[]>(hpx::find_here(), 10, ...);

hpx::id_type id = f.get();
```

**Return** The function returns different types depending on its use:
- If the explicit template argument `Component` represents an array of a component type (i.e. `Component[]`, where `traits::is_component<Component>::value` evaluates to true), the function will return an `hpx:future` object instance which holds a `std::vector<hpx::id_type>`, where each of the items in this vector is a global address of one of the newly created components.
- If the explicit template argument `Component` represents an array of a client side object type (i.e. `Component[]`, where `traits::is_client<Component>::value` evaluates to true), the function will return an `hpx:future` object instance which holds a `std::vector<hpx::id_type>`, where each of the items in this vector is a client side instance of the given type, each representing one of the newly created components.

**Parameters**
- `locality`: [in] The global address of the locality where the new instance should be created on.
- `count`: [in] The number of component instances to create
- `vs`: [in] Any number of arbitrary arguments (passed by value, by const reference or by rvalue reference) which will be forwarded to the constructor of the created component instance.

```cpp
template <typename Component, typename typename DistPolicy, typename typename ... Ts>
<unspecified> hpx::new_(DistPolicy const & policy, Ts &&... vs)
```

Create one or more new instances of the given Component type based on the given distribution policy.

This function creates one or more new instances of the given Component type on the localities defined by the given distribution policy and returns a future object for global address which can be used to reference the new component instance(s).

**Note** This function requires to specify an explicit template argument which will define what type of component(s) to create, for instance:

```cpp
hpx::future<hpx::id_type> f =
  hpx::new_<some_component>(hpx::default_layout, ...);

hpx::id_type id = f.get();
```

**Return** The function returns different types depending on its use:
• If the explicit template argument Component represents a component type (traits::is_component<Component>::value evaluates to true), the function will return an hpx::future object instance which can be used to retrieve the global address of the newly created component.

• If the explicit template argument Component represents a client side object (traits::is_client<Component>::value evaluates to true), the function will return a new instance of that type which can be used to refer to the newly created component instance.

Parameters

• policy: [in] The distribution policy used to decide where to place the newly created.

• vs: [in] Any number of arbitrary arguments (passed by value, by const reference or by rvalue reference) which will be forwarded to the constructor of the created component instance.

template <typename Component, typename DistPolicy, typename ... Ts>
<hpx::future<std::vector<hpx::id_type>> > f =
    hpx::new_<some_component[]>(hpx::default_layout, 10, ...);
    hpx::id_type id = f.get();

Return The function returns different types depending on its use:

• If the explicit template argument Component represents an array of a component type (i.e. Component[], where traits::is_component<Component>::value evaluates to true), the function will return an hpx::future object instance which holds a std::vector<hpx::id_type>, where each of the items in this vector is a global address of one of the newly created components.

• If the explicit template argument Component represents an array of a client side object type (i.e. Component[], where traits::is_client<Component>::value evaluates to true), the function will return an hpx::future object instance which holds a std::vector<hpx::id_type>, where each of the items in this vector is a client side instance of the given type, each representing one of the newly created components.

Parameters

• policy: [in] The distribution policy used to decide where to place the newly created.

• count: [in] The number of component instances to create

• vs: [in] Any number of arbitrary arguments (passed by value, by const reference or by rvalue reference) which will be forwarded to the constructor of the created component instance.

template <typename ... Ts>
tuple<hpx::future<Ts>> split_future (future<hpx::future<Ts>> &f)

The function split_future is an operator allowing to split a given future of a sequence of values (any tuple, std::pair, or std::array) into an equivalent container of futures where each future represents one of the values from the original future. In some sense this function provides the inverse operation of when_all.
Return Returns an equivalent container (same container type as passed as the argument) of futures, where each future refers to the corresponding value in the input parameter. All of the returned futures become ready once the input future has become ready. If the input future is exceptional, all output futures will be exceptional as well.

Note The following cases are special:

```
tuple<future<void> > split_future(future<tuple<> > & & f);
array<future<void>, 1> split_future(future<array<T, 0>, 0> & & f);
```

here the returned futures are directly representing the futures which were passed to the function.

Parameters

- **f**: [in] A future holding an arbitrary sequence of values stored in a tuple-like container. This facility supports `hp::util::tuple<>`, `std::pair<T1, T2>`, and `std::array<T, N>`

```
template <typename T>
std::vector<future<T>> split_future (future<std::vector<T>> & & f, std::size_t size)
```

The function `split_future` is an operator allowing to split a given future of a sequence of values (any std::vector) into a std::vector of futures where each future represents one of the values from the original std::vector. In some sense this function provides the inverse operation of `when_all`.

Return Returns a std::vector of futures, where each future refers to the corresponding value in the input parameter. All of the returned futures become ready once the input future has become ready. If the input future is exceptional, all output futures will be exceptional as well.

Parameters

- **f**: [in] A future holding an arbitrary sequence of values stored in a std::vector.
- **size**: [in] The number of elements the vector will hold once the input future has become ready

```
template <typename InputIter>
void wait_all (InputIter first, InputIter last)
```

The function `wait_all` is an operator allowing to join on the result of all given futures. It AND-composes all future objects given and returns after they finished executing.

Note The function `wait_all` returns after all futures have become ready. All input futures are still valid after `wait_all` returns.

Parameters

- **first**: The iterator pointing to the first element of a sequence of `future` or `shared_future` objects for which `wait_all` should wait.
- **last**: The iterator pointing to the last element of a sequence of `future` or `shared_future` objects for which `wait_all` should wait.

```
template <typename R>
void wait_all (std::vector<future<R>> & & futures)
```

The function `wait_all` is an operator allowing to join on the result of all given futures. It AND-composes all future objects given and returns after they finished executing.

Note The function `wait_all` returns after all futures have become ready. All input futures are still valid after `wait_all` returns.

Parameters
• futures: A vector or array holding an arbitrary amount of future or shared_future objects for which wait_all should wait.

```cpp
template<typename R, std::size_t N>
void wait_all(std::array<future<R>, N> &&futures)
```

The function `wait_all` is an operator allowing to join on the result of all given futures. It AND-composes all future objects given and returns after they finished executing.

**Note** The function `wait_all` returns after all futures have become ready. All input futures are still valid after `wait_all` returns.

**Parameters**

- futures: A vector or array holding an arbitrary amount of future or shared_future objects for which `wait_all` should wait.

```cpp
template<typename ... T>
void wait_all(T&&... futures)
```

The function `wait_all` is an operator allowing to join on the result of all given futures. It AND-composes all future objects given and returns after they finished executing.

**Note** The function `wait_all` returns after all futures have become ready. All input futures are still valid after `wait_all` returns.

**Parameters**

- futures: An arbitrary number of future or shared_future objects, possibly holding different types for which `wait_all` should wait.

```cpp
template<typename InputIter>
InputIter wait_all_n(InputIter begin, std::size_t count)
```

The function `wait_all_n` is an operator allowing to join on the result of all given futures. It AND-composes all future objects given and returns after they finished executing.

**Return** The function `wait_all_n` will return an iterator referring to the first element in the input sequence after the last processed element.

**Note** The function `wait_all_n` returns after all futures have become ready. All input futures are still valid after `wait_all_n` returns.

**Parameters**

- begin: The iterator pointing to the first element of a sequence of future or shared_future objects for which `wait_all_n` should wait.
- count: The number of elements in the sequence starting at `first`.

```cpp
template<typename InputIter, typename Container = vector<future<typename std::iterator_traits<InputIter>::value_type>>> future<Container> when_all(InputIter first, InputIter last)
```

The function `when_all` is an operator allowing to join on the result of all given futures. It AND-composes all future objects given and returns a new future object representing the same list of futures after they finished executing.

**Return** Returns a future holding the same list of futures as has been passed to `when_all`.

- future<Container<future<R>>>: If the input cardinality is unknown at compile time and the futures are all of the same type. The order of the futures in the output container will be the same as given by the input iterator.
**Note** Calling this version of *when_all* where first == last, returns a future with an empty container that is immediately ready. Each future and shared_future is waited upon and then copied into the collection of the output (returned) future, maintaining the order of the futures in the input collection. The future returned by *when_all* will not throw an exception, but the futures held in the output collection may.

**Parameters**

- **first**: [in] The iterator pointing to the first element of a sequence of *future* or *shared_future* objects for which *when_all* should wait.
- **last**: [in] The iterator pointing to the last element of a sequence of *future* or *shared_future* objects for which *when_all* should wait.

```template <typename Range>
future<Range> when_all(Range &&values)
```

The function *when_all* is an operator allowing to join on the result of all given futures. It AND-composes all future objects given and returns a new future object representing the same list of futures after they finished executing.

**Return** Returns a future holding the same list of futures as has been passed to *when_all*.

- **future<Container<future<R>>>**: If the input cardinality is unknown at compile time and the futures are all of the same type.

**Note** Calling this version of *when_all* where the input container is empty, returns a future with an empty container that is immediately ready. Each future and shared_future is waited upon and then copied into the collection of the output (returned) future, maintaining the order of the futures in the input collection. The future returned by *when_all* will not throw an exception, but the futures held in the output collection may.

**Parameters**

- **values**: [in] A range holding an arbitrary amount of *future* or *shared_future* objects for which *when_all* should wait.

```template <typename ... T>
future<tuple<future<T>...>> when_all(T&&... futures)
```

The function *when_all* is an operator allowing to join on the result of all given futures. It AND-composes all future objects given and returns a new future object representing the same list of futures after they finished executing.

**Return** Returns a future holding the same list of futures as has been passed to *when_all*.

- **future<tuple<future<T0>, future<T1>, future<T2>...>>**: If inputs are fixed in number and are of heterogeneous types. The inputs can be any arbitrary number of future objects.
- **future<tuple<> if *when_all* is called with zero arguments. The returned future will be initially ready.**

**Note** Each future and shared_future is waited upon and then copied into the collection of the output (returned) future, maintaining the order of the futures in the input collection. The future returned by *when_all* will not throw an exception, but the futures held in the output collection may.

**Parameters**

- **futures**: [in] An arbitrary number of *future* or *shared_future* objects, possibly holding different types for which *when_all* should wait.
future<Container> when_all_n (InputIter begin, std::size_t count)

The function when_all_n is an operator allowing to join on the result of all given futures. It AND-composes all future objects given and returns a new future object representing the same list of futures after they finished executing.

Return
Returns a future holding the same list of futures as has been passed to when_all_n.

- future<Container<future<R>>>: If the input cardinality is unknown at compile time and the futures are all of the same type. The order of the futures in the output vector will be the same as given by the input iterator.

Note
As long as ec is not pre-initialized to hpx::throws this function doesn’t throw but returns the result code using the parameter ec. Otherwise it throws an instance of hpx::exception.

Note
None of the futures in the input sequence are invalidated.

Parameters

- begin: [in] The iterator pointing to the first element of a sequence of future or shared_future objects for which wait_all_n should wait.
- count: [in] The number of elements in the sequence starting at first.

Exceptions

- This: function will throw errors which are encountered while setting up the requested operation only. Errors encountered while executing the operations delivering the results to be stored in the futures are reported through the futures themselves.

template <typename InputIter>
void wait_any (InputIter first, InputIter last, error_code &ec = throws)

The function wait_any is a non-deterministic choice operator. It OR-composes all future objects given and returns after one future of that list finishes execution.

Note
The function wait_any returns after at least one future has become ready. All input futures are still valid after wait_any returns.

Note
As long as ec is not pre-initialized to hpx::throws this function doesn’t throw but returns the result code using the parameter ec. Otherwise it throws an instance of hpx::exception.

Note
None of the futures in the input sequence are invalidated.

Parameters

- first: [in] The iterator pointing to the first element of a sequence of future or shared_future objects for which wait_any should wait.
- last: [in] The iterator pointing to the last element of a sequence of future or shared_future objects for which wait_any should wait.
- ec: [in,out] this represents the error status on exit, if this is pre-initialized to hpx::throws the function will throw on error instead.

template <typename R>
void wait_any (std::vector<future<R>> &futures, error_code &ec = throws)

The function wait_any is a non-deterministic choice operator. It OR-composes all future objects given and returns after one future of that list finishes execution.

Note
The function wait_any returns after at least one future has become ready. All input futures are still valid after wait_any returns.
**Note** As long as `ec` is not pre-initialized to `hpx::throws` this function doesn’t throw but returns the result code using the parameter `ec`. Otherwise it throws an instance of `hpx::exception`.

**Note** None of the futures in the input sequence are invalidated.

**Parameters**
- `futures`: [in] A vector holding an arbitrary amount of `future` or `shared_future` objects for which `wait_any` should wait.
- `ec`: [in,out] this represents the error status on exit, if this is pre-initialized to `hpx::throws` the function will throw on error instead.

```cpp
template <typename R, std::size_t N>
void wait_any(std::array<future<R>, N> &futures, error_code &ec = throws)
```

The function `wait_any` is a non-deterministic choice operator. It OR-composes all future objects given and returns after one future of that list finishes execution.

**Note** The function `wait_any` returns after at least one future has become ready. All input futures are still valid after `wait_any` returns.

**Note** As long as `ec` is not pre-initialized to `hpx::throws` this function doesn’t throw but returns the result code using the parameter `ec`. Otherwise it throws an instance of `hpx::exception`.

**Note** None of the futures in the input sequence are invalidated.

**Parameters**
- `futures`: [in] An array holding an arbitrary amount of `future` or `shared_future` objects for which `wait_any` should wait.
- `ec`: [in,out] this represents the error status on exit, if this is pre-initialized to `hpx::throws` the function will throw on error instead.

```cpp
template <typename ... T>
void wait_any(error_code &ec, T&&... futures)
```

The function `wait_any` is a non-deterministic choice operator. It OR-composes all future objects given and returns after one future of that list finishes execution.

**Note** The function `wait_any` returns after at least one future has become ready. All input futures are still valid after `wait_any` returns.

**Note** As long as `ec` is not pre-initialized to `hpx::throws` this function doesn’t throw but returns the result code using the parameter `ec`. Otherwise it throws an instance of `hpx::exception`.

**Note** None of the futures in the input sequence are invalidated.

**Parameters**
- `futures`: [in] An arbitrary number of `future` or `shared_future` objects, possibly holding different types for which `wait_any` should wait.
- `ec`: [in,out] this represents the error status on exit, if this is pre-initialized to `hpx::throws` the function will throw on error instead.
Note The function `wait_any` returns after at least one future has become ready. All input futures are still valid after `wait_any` returns.

Note None of the futures in the input sequence are invalidated.

Parameters

- `futures`: [in] An arbitrary number of `future` or `shared_future` objects, possibly holding different types for which `wait_any` should wait.

```
template <typename InputIter>
InputIter wait_any_n (InputIter first, std::size_t count, error_code &ec = throws)
```

The function `wait_any_n` is a non-deterministic choice operator. It OR-composes all future objects given and returns after one future of that list finishes execution.

Note The function `wait_any_n` returns after at least one future has become ready. All input futures are still valid after `wait_any_n` returns.

Return The function `wait_all_n` will return an iterator referring to the first element in the input sequence after the last processed element.

Note As long as `ec` is not pre-initialized to `hpx::throws` this function doesn’t throw but returns the result code using the parameter `ec`. Otherwise it throws an instance of `hpx::exception`.

Note None of the futures in the input sequence are invalidated.

Parameters

- `first`: [in] The iterator pointing to the first element of a sequence of `future` or `shared_future` objects for which `wait_any_n` should wait.
- `count`: [in] The number of elements in the sequence starting at `first`.
- `ec`: [in,out] this represents the error status on exit, if this is pre-initialized to `hpx::throws` the function will throw on error instead.

```
template <typename InputIter, typename Container = vector<future<typename std::iterator_traits<InputIter>::value_type>>> future<when_any_result<Container>> when_any (InputIter first, InputIter last)
```

The function `when_any` is a non-deterministic choice operator. It OR-composes all future objects given and returns a new future object representing the same list of futures after one future of that list finishes execution.

Return Returns a `when_any_result` holding the same list of futures as has been passed to `when_any` and an index pointing to a ready future.

- `future<when_any_result<Container<future<R>>>`: If the input cardinality is unknown at compile time and the futures are all of the same type. The order of the futures in the output container will be the same as given by the input iterator.

Parameters

- `first`: [in] The iterator pointing to the first element of a sequence of `future` or `shared_future` objects for which `when_any` should wait.
- `last`: [in] The iterator pointing to the last element of a sequence of `future` or `shared_future` objects for which `when_any` should wait.

```
template <typename Range> future<when_any_result<Range>> when_any (Range &values)
```

The function `when_any` is a non-deterministic choice operator. It OR-composes all future objects given
and returns a new future object representing the same list of futures after one future of that list finishes execution.

**Return** Returns a `when_any_result` holding the same list of futures as has been passed to `when_any` and an index pointing to a ready future.

- `future<when_any_result<Container<future<R>>>>`: If the input cardinality is unknown at compile time and the futures are all of the same type. The order of the futures in the output container will be the same as given by the input iterator.

**Parameters**

- `values`: [in] A range holding an arbitrary amount of `future` or `shared_future` objects for which `when_any` should wait.

```template<typename ...T>
future<when_any_result<tuple<future<T>...>> when_any(T&&... futures)
```

The function `when_any` is a non-deterministic choice operator. It OR-composes all future objects given and returns a new future object representing the same list of futures after one future of that list finishes execution.

**Return** Returns a `when_any_result` holding the same list of futures as has been passed to `when_any` and an index pointing to a ready future.

- `future<when_any_result<tuple<future<T0>, future<T1>...>>`: If inputs are fixed in number and are of heterogeneous types. The inputs can be any arbitrary number of future objects.

- `future<when_any_result<tuple<>>>` if `when_any` is called with zero arguments. The returned future will be initially ready.

**Parameters**

- `futures`: [in] An arbitrary number of `future` or `shared_future` objects, possibly holding different types for which `when_any` should wait.

```template<typename InputIter, typename Container = vector<future<typename std::iterator_traits<InputIter>::value_type>>
future<when_any_result<Container>> when_any_n(InputIter first, std::size_t count)
```

The function `when_any_n` is a non-deterministic choice operator. It OR-composes all future objects given and returns a new future object representing the same list of futures after one future of that list finishes execution.

**Return** Returns a `when_any_result` holding the same list of futures as has been passed to `when_any` and an index pointing to a ready future.

- `future<when_any_result<Container<future<R>>>>`: If the input cardinality is unknown at compile time and the futures are all of the same type. The order of the futures in the output container will be the same as given by the input iterator.

**Note** None of the futures in the input sequence are invalidated.

**Parameters**

- `first`: [in] The iterator pointing to the first element of a sequence of `future` or `shared_future` objects for which `when_any_n` should wait.

- `count`: [in] The number of elements in the sequence starting at `first`.

```template<typename InputIter>
future<vector<future<typename std::iterator_traits<InputIter>::value_type>>> wait_some (std::size_t n, Iterator first, Iterator last, error_code &ec = throws)

The function wait_some is an operator allowing to join on the result of all given futures. It AND-composes all future objects given and returns a new future object representing the same list of futures after n of them finished executing.

**Note** The future returned by the function wait_some becomes ready when at least n argument futures have become ready.

**Return** Returns a future holding the same list of futures as has been passed to wait_some.
- future<vector<future<R>>>: If the input cardinality is unknown at compile time and the futures are all of the same type.

**Note** Calling this version of wait_some where first == last, returns a future with an empty vector that is immediately ready. Each future and shared_future is waited upon and then copied into the collection of the output (returned) future, maintaining the order of the futures in the input collection. The future returned by wait_some will not throw an exception, but the futures held in the output collection may.

**Parameters**
- n: [in] The number of futures out of the arguments which have to become ready in order for the returned future to get ready.
- first: [in] The iterator pointing to the first element of a sequence of future or shared_future objects for which when_all should wait.
- last: [in] The iterator pointing to the last element of a sequence of future or shared_future objects for which when_all should wait.
- ec: [in,out] this represents the error status on exit, if this is pre-initialized to hpx::throws the function will throw on error instead.

**template <typename R>**

void wait_some (std::size_t n, std::vector<future<R>> &&futures, error_code &ec = throws)

The function wait_all is an operator allowing to join on the result of all given futures. It AND-composes all future objects given and returns a new future object representing the same list of futures after n of them finished executing.

**Note** The function wait_all returns after n futures have become ready. All input futures are still valid after wait_all returns.

**Note** Each future and shared_future is waited upon and then copied into the collection of the output (returned) future, maintaining the order of the futures in the input collection. The future returned by wait_some will not throw an exception, but the futures held in the output collection may.

**Parameters**
- n: [in] The number of futures out of the arguments which have to become ready in order for the returned future to get ready.
- futures: [in] A vector holding an arbitrary amount of future or shared_future objects for which wait_some should wait.
• ec: [in,out] this represents the error status on exit, if this is pre-initialized to `hpx::throws` the function will throw on error instead.

```cpp
template <typename R, std::size_t N>
void wait_some(std::size_t n, std::array<future<R>, N> &&futures, error_code &ec = throws)
```

The function `wait_some` is an operator allowing to join on the result of all given futures. It AND-composes all future objects given and returns a new future object representing the same list of futures after `n` of them finished executing.

**Note** The function `wait_all` returns after `n` futures have become ready. All input futures are still valid after `wait_all` returns.

**Note** Each future and shared_future is waited upon and then copied into the collection of the output (returned) future, maintaining the order of the futures in the input collection. The future returned by `wait_some` will not throw an exception, but the futures held in the output collection may.

**Parameters**

- `n`: [in] The number of futures out of the arguments which have to become ready in order for the returned future to get ready.
- `futures`: [in] An array holding an arbitrary amount of `future` or `shared_future` objects for which `wait_some` should wait.
- `ec`: [in,out] this represents the error status on exit, if this is pre-initialized to `hpx::throws` the function will throw on error instead.

```cpp
template <typename ... T>
void wait_some(std::size_t n, Iterator first, std::size_t count, error_code &ec = throws)
```

The function `wait_some` is an operator allowing to join on the result of all given futures. It AND-composes all future objects given and returns a new future object representing the same list of futures after `n` of them finished executing.

**Note** The function `wait_all` returns after `n` futures have become ready. All input futures are still valid after `wait_all` returns.

**Note** Calling this version of `wait_some` where `first` == `last`, returns a future with an empty vector that is immediately ready. Each future and shared_future is waited upon and then copied into the collection of the output (returned) future, maintaining the order of the futures in the input collection. The future returned by `wait_some` will not throw an exception, but the futures held in the output collection may.

**Parameters**

- `n`: [in] The number of futures out of the arguments which have to become ready in order for the returned future to get ready.
- `futures`: [in] An arbitrary number of `future` or `shared_future` objects, possibly holding different types for which `wait_some` should wait.
- `ec`: [in,out] this represents the error status on exit, if this is pre-initialized to `hpx::throws` the function will throw on error instead.
The function `wait_all` returns after \( n \) futures have become ready. All input futures are still valid after `wait_all` returns.

This function returns an Iterator referring to the first element after the last processed input element.

Calling this version of `wait_some_n` where `count` == 0, returns a future with the same elements as the arguments that is immediately ready. Possibly none of the futures in that vector are ready. Each future and shared_future is waited upon and then copied into the collection of the output (returned) future, maintaining the order of the futures in the input collection. The future returned by `wait_some_n` will not throw an exception, but the futures held in the output collection may.

**Parameters**

- \( n \): [in] The number of futures out of the arguments which have to become ready in order for the returned future to get ready.
- `first`: [in] The iterator pointing to the first element of a sequence of `future` or `shared_future` objects for which `when_all` should wait.
- `count`: [in] The number of elements in the sequence starting at `first`.
- `ec`: [in,out] this represents the error status on exit, if this is pre-initialized to `hpx::throws` the function will throw on error instead.

```
template <typename InputIter, typename Container = vector<future<typename std::iterator_traits<InputIter>::value_type>>> future<when_some_result<Container>> when_some (std::size_t n, InputIter first, InputIter last, error_code &ec = throws)
```

The function `when_some` is an operator allowing to join on the result of all given futures. It AND-composes all future objects given and returns a new future object representing the same list of futures after \( n \) of them finished executing.

The future returned by the function `when_some` becomes ready when at least \( n \) argument futures have become ready.

**Return** Returns a `when_some_result` holding the same list of futures as has been passed to `when_some` and indices pointing to ready futures.

- `future<when_some_result<Container<future<R>>>`: If the input cardinality is unknown at compile time and the futures are all of the same type. The order of the futures in the output container will be the same as given by the input iterator.

Calling this version of `when_some` where `first` == `last`, returns a future with an empty container that is immediately ready. Each future and shared_future is waited upon and then copied into the collection of the output (returned) future, maintaining the order of the futures in the input collection. The future returned by `when_some` will not throw an exception, but the futures held in the output collection may.

**Parameters**

- \( n \): [in] The number of futures out of the arguments which have to become ready in order for the returned future to get ready.
- `first`: [in] The iterator pointing to the first element of a sequence of `future` or `shared_future` objects for which `when_all` should wait.
- `last`: [in] The iterator pointing to the last element of a sequence of `future` or `shared_future` objects for which `when_all` should wait.
- `ec`: [in,out] this represents the error status on exit, if this is pre-initialized to `hpx::throws` the function will throw on error instead.

```
template <typename Range>
```
future<when_some_result<Range>> when_some (std::size_t n, Range &&futures, error_code &ec = throws)

The function when_some is an operator allowing to join on the result of all given futures. It AND-composes all future objects given and returns a new future object representing the same list of futures after \( n \) of them finished executing.

**Note** The future returned by the function when_some becomes ready when at least \( n \) argument futures have become ready.

**Return** Returns a when_some_result holding the same list of futures as has been passed to when_some and indices pointing to ready futures.

- future<when_some_result<Container<future<R>>>>: If the input cardinality is unknown at compile time and the futures are all of the same type. The order of the futures in the output container will be the same as given by the input iterator.

**Note** Each future and shared_future is waited upon and then copied into the collection of the output (returned) future, maintaining the order of the futures in the input collection. The future returned by when_some will not throw an exception, but the futures held in the output collection may.

**Parameters**

- \( n \): [in] The number of futures out of the arguments which have to become ready in order for the returned future to get ready.
- futures: [in] A container holding an arbitrary amount of future or shared_future objects for which when_some should wait.
- ec: [in,out] this represents the error status on exit, if this is pre-initialized to hpx::throws the function will throw on error instead.

`template <typename ... T>`

future<when_some_result<tuple<future<T>...>>> when_some (std::size_t n, error_code &ec, T&&...

futures)

The function when_some is an operator allowing to join on the result of all given futures. It AND-composes all future objects given and returns a new future object representing the same list of futures after \( n \) of them finished executing.

**Note** The future returned by the function when_some becomes ready when at least \( n \) argument futures have become ready.

**Return** Returns a when_some_result holding the same list of futures as has been passed to when_some and an index pointing to a ready future.

- future<when_some_result<tuple<future<T0>, future<T1>...>>>: If inputs are fixed in number and are of heterogeneous types. The inputs can be any arbitrary number of future objects.
- future<when_some_result<tuple<>>> if when_some is called with zero arguments. The returned future will be initially ready.

**Note** Each future and shared_future is waited upon and then copied into the collection of the output (returned) future, maintaining the order of the futures in the input collection. The future returned by when_some will not throw an exception, but the futures held in the output collection may.

**Parameters**

- \( n \): [in] The number of futures out of the arguments which have to become ready in order for the returned future to get ready.
- ec: [in,out] this represents the error status on exit, if this is pre-initialized to hpx::throws the function will throw on error instead.
futures: [in] An arbitrary number of future or shared_future objects, possibly holding different types for which when_some should wait.

```cpp
template <typename ... T>
future<when_some_result<tuple<futurer<T>...>> when_some (std::size_t n, T&... futures)
```

The function when_some is an operator allowing to join on the result of all given futures. It AND-composes all future objects given and returns a new future object representing the same list of futures after n of them finished executing.

**Note** The future returned by the function when_some becomes ready when at least n argument futures have become ready.

**Return** Returns a when_some_result holding the same list of futures as has been passed to when_some and an index pointing to a ready future.

- future<when_some_result<tuple<futurer<T0>, futurer<T1>, ...>>: If inputs are fixed in number and are of heterogeneous types. The inputs can be any arbitrary number of future objects.
- future<when_some_result<tuple<>>> if when_some is called with zero arguments. The returned future will be initially ready.

**Note** Each future and shared_future is waited upon and then copied into the collection of the output (returned) future, maintaining the order of the futures in the input collection. The future returned by when_some will not throw an exception, but the futures held in the output collection may.

**Parameters**

- n: [in] The number of futures out of the arguments which have to become ready in order for the returned future to get ready.
- futures: [in] An arbitrary number of future or shared_future objects, possibly holding different types for which when_some should wait.

```cpp
template <typename InputIter, typename Container = vector<future<typename std::iterator_traits<InputIter>::value_type>>> future<when_some_result<Container>> when_some_n (std::size_t n, iterator first, std::size_t count, error_code &ec = throws)
```

The function when_some_n is an operator allowing to join on the result of all given futures. It AND-composes all future objects given and returns a new future object representing the same list of futures after n of them finished executing.

**Note** The future returned by the function when_some_n becomes ready when at least n argument futures have become ready.

**Return** Returns a when_some_result holding the same list of futures as has been passed to when_some and indices pointing to ready futures.

- future<when_some_result<Container<future<R>>>: If the input cardinality is unknown at compile time and the futures are all of the same type. The order of the futures in the output container will be the same as given by the input iterator.

**Note** Calling this version of when_some_n where count == 0, returns a future with the same elements as the arguments that is immediately ready. Possibly none of the futures in that container are ready. Each future and shared_future is waited upon and then copied into the collection of the output (returned) future, maintaining the order of the futures in the input collection. The future returned by when_some_n will not throw an exception, but the futures held in the output collection may.

**Parameters**

- n: [in] The number of futures out of the arguments which have to become ready in order for the returned future to get ready.
HPX Documentation, 1.4.1

- **first**: [in] The iterator pointing to the first element of a sequence of *future* or *shared_future* objects for which *when_all* should wait.
- **count**: [in] The number of elements in the sequence starting at first.
- **ec**: [in,out] this represents the error status on exit, if this is pre-initialized to `hpx::throws` the function will throw on error instead.

`template <typename F, typename Future> void wait_each (F && f, std::vector<Future> && futures)`

The function `wait_each` is an operator allowing to join on the results of all given futures. It AND-composes all future objects given and returns after they finished executing. Additionally, the supplied function is called for each of the passed futures as soon as the future has become ready. `wait_each` returns after all futures have been become ready.

**Note** This function consumes the futures as they are passed on to the supplied function. The callback should take one or two parameters, namely either a *future* to be processed or a type that `std::size_t` is implicitly convertible to as the first parameter and the *future* as the second parameter. The first parameter will correspond to the index of the current *future* in the collection.

**Parameters**

- **f**: The function which will be called for each of the input futures once the future has become ready.
- **futures**: A vector holding an arbitrary amount of *future* or *shared_future* objects for which `wait_each` should wait.

`template <typename F, typename Iterator> void wait_each (F && f, Iterator begin, Iterator end)`

The function `wait_each` is an operator allowing to join on the results of all given futures. It AND-composes all future objects given and returns after they finished executing. Additionally, the supplied function is called for each of the passed futures as soon as the future has become ready. `wait_each` returns after all futures have been become ready.

**Note** This function consumes the futures as they are passed on to the supplied function. The callback should take one or two parameters, namely either a *future* to be processed or a type that `std::size_t` is implicitly convertible to as the first parameter and the *future* as the second parameter. The first parameter will correspond to the index of the current *future* in the collection.

**Parameters**

- **f**: The function which will be called for each of the input futures once the future has become ready.
- **begin**: The iterator pointing to the first element of a sequence of *future* or *shared_future* objects for which `wait_each` should wait.
- **end**: The iterator pointing to the last element of a sequence of *future* or *shared_future* objects for which `wait_each` should wait.

`template <typename F, typename ... T> void wait_each (F && f, T&&... futures)`

The function `wait_each` is an operator allowing to join on the results of all given futures. It AND-composes all future objects given and returns after they finished executing. Additionally, the supplied function is called for each of the passed futures as soon as the future has become ready. `wait_each` returns after all futures have been become ready.
**Note** This function consumes the futures as they are passed on to the supplied function. The callback should take one or two parameters, namely either a `future` to be processed or a type that `std::size_t` is implicitly convertible to as the first parameter and the `future` as the second parameter. The first parameter will correspond to the index of the current `future` in the collection.

**Parameters**

- **f**: The function which will be called for each of the input futures once the future has become ready.
- **futures**: An arbitrary number of `future` or `shared_future` objects, possibly holding different types for which `wait_each` should wait.

```cpp
template <typename F, typename Iterator>
void wait_each_n(F &&f, Iterator begin, std::size_t count)
```

The function `wait_each` is an operator allowing to join on the result of all given futures. It AND-composes all future objects given and returns after they finished executing. Additionally, the supplied function is called for each of the passed futures as soon as the future has become ready.

**Note** This function consumes the futures as they are passed on to the supplied function. The callback should take one or two parameters, namely either a `future` to be processed or a type that `std::size_t` is implicitly convertible to as the first parameter and the `future` as the second parameter. The first parameter will correspond to the index of the current `future` in the collection.

**Parameters**

- **f**: The function which will be called for each of the input futures once the future has become ready.
- **begin**: The iterator pointing to the first element of a sequence of `future` or `shared_future` objects for which `wait_each_n` should wait.
- **count**: The number of elements in the sequence starting at `first`.

```cpp
template <typename F, typename Future>
future<void> when_each(F &&f, std::vector<Future> &&futures)
```

The function `when_each` is an operator allowing to join on the results of all given futures. It AND-composes all future objects given and returns a new future object representing the event of all those futures having finished executing. It also calls the supplied callback for each of the futures which becomes ready.

**Note** This function consumes the futures as they are passed on to the supplied function. The callback should take one or two parameters, namely either a `future` to be processed or a type that `std::size_t` is implicitly convertible to as the first parameter and the `future` as the second parameter. The first parameter will correspond to the index of the current `future` in the collection.

**Return** Returns a future representing the event of all input futures being ready.

**Parameters**

- **f**: The function which will be called for each of the input futures once the future has become ready.
- **futures**: A vector holding an arbitrary amount of `future` or `shared_future` objects for which `wait_each` should wait.

```cpp
template <typename F, typename Iterator>
```
future<Iterator> when_each (F &&f, Iterator begin, Iterator end)

The function when_each is an operator allowing to join on the results of all given futures. It AND-composes all future objects given and returns a new future object representing the event of all those futures having finished executing. It also calls the supplied callback for each of the futures which becomes ready.

Note This function consumes the futures as they are passed on to the supplied function. The callback should take one or two parameters, namely either a future to be processed or a type that \( \text{std::size_t} \) is implicitly convertible to as the first parameter and the future as the second parameter. The first parameter will correspond to the index of the current future in the collection.

Return Returns a future representing the event of all input futures being ready.

Parameters

- f: The function which will be called for each of the input futures once the future has become ready.
- begin: The iterator pointing to the first element of a sequence of future or shared_future objects for which wait_each should wait.
- end: The iterator pointing to the last element of a sequence of future or shared_future objects for which wait_each should wait.

\[
\text{template <typename } F, \text{ typename... } Ts> \text{ future<Iterator> when_each_n (F &&f, Iterator begin, std::size_t count)}
\]

The function when_each is an operator allowing to join on the results of all given futures. It AND-composes all future objects given and returns a new future object representing the event of all those futures having finished executing. It also calls the supplied callback for each of the futures which becomes ready.

Note This function consumes the futures as they are passed on to the supplied function. The callback should take one or two parameters, namely either a future to be processed or a type that \( \text{std::size_t} \) is implicitly convertible to as the first parameter and the future as the second parameter. The first parameter will correspond to the index of the current future in the collection.

Return Returns a future holding the iterator pointing to the first element after the last one.

Parameters

- f: The function which will be called for each of the input futures once the future has become ready.
- futures: An arbitrary number of future or shared_future objects, possibly holding different types for which wait_each should wait.
• f: The function which will be called for each of the input futures once the future has become ready.

• begin: The iterator pointing to the first element of a sequence of future or shared_future objects for which wait_each_n should wait.

• count: The number of elements in the sequence starting at first.

namespace actions
namespace applier

Functions

applier & get_applier()
The function get_applier returns a reference to the (thread specific) applier instance.

applier * get_applier_ptr()
The function get_applier returns a pointer to the (thread specific) applier instance. The returned pointer is NULL if the current thread is not known to HPX or if the runtime system is not active.

namespace components

Functions

template <typename Component>
future<naming::id_type> migrate_from_storage(naming::id_type const &to_resurrect, naming::id_type const &target = naming::invalid_id)
Migrate the component with the given id from the specified target storage (resurrect the object)
The function migrate_from_storage<Component> will migrate the component referenced by to_resurrect from the storage facility specified where the object is currently stored on. It returns a future referring to the migrated component instance. The component instance is resurrected on the locality specified by target_locality.

Return A future representing the global id of the migrated component instance. This should be the same as to_resurrect.

Parameters

• to_resurrect: [in] The global id of the component to migrate.

• target: [in] The optional locality to resurrect the object on. By default the object is resurrected on the locality it was located on last.

Template Parameters

• The: only template argument specifies the component type of the component to migrate from the given storage facility.

template <typename Component>
future<naming::id_type> migrate_to_storage(naming::id_type const &to_migrate, naming::id_type const &target_storage)
Migrate the component with the given id to the specified target storage
The function migrate_to_storage<Component> will migrate the component referenced by to_migrate to the storage facility specified with target_storage. It returns a future referring to the migrated component instance.
Return A future representing the global id of the migrated component instance. This should be the same as `migrate_to`.

Parameters
- `to_migrate`: [in] The global id of the component to migrate.
- `target_storage`: [in] The id of the storage facility to migrate this object to.

Template Parameters
- `The`: only template argument specifies the component type of the component to migrate to the given storage facility.

```template <typename Derived, typename Stub>
Derived migrate_to_storage (client_base<Derived, Stub> const &to_migrate, hpx::components::component_storage const &target_storage)
```

Migrate the given component to the specified target storage.

The function `migrate_to_storage` will migrate the component referenced by `to_migrate` to the storage facility specified with `target_storage`. It returns a future referring to the migrated component instance.

Return A client side representation of representing of the migrated component instance. This should be the same as `migrate_to`.

Parameters
- `to_migrate`: [in] The client side representation of the component to migrate.
- `target_storage`: [in] The id of the storage facility to migrate this object to.

```template <typename Component>
future<naming::id_type> copy (naming::id_type const &to_copy)
```

Copy given component to the specified target locality.

The function `copy<Component>` will create a copy of the component referenced by `to_copy` on the locality specified with `target_locality`. It returns a future referring to the newly created component instance.

Return A future representing the global id of the newly (copied) component instance.

Note The new component instance is created on the locality of the component instance which is to be copied.

Parameters
- `to_copy`: [in] The global id of the component to copy

Template Parameters
- `The`: only template argument specifies the component type to create.

```template <typename Component>
future<naming::id_type> copy (naming::id_type const &to_copy, naming::id_type const &target_locality)
```

Copy given component to the specified target locality.

The function `copy<Component>` will create a copy of the component referenced by `to_copy` on the locality specified with `target_locality`. It returns a future referring to the newly created component instance.

Return A future representing the global id of the newly (copied) component instance.

Parameters
• to_copy: [in] The global id of the component to copy
• target_locality: [in] The locality where the copy should be created.

Template Parameters
• The: only template argument specifies the component type to create.

template <typename Derived, typename Stub>
Derived copy (client_base<Derived, Stub> const &to_copy, naming::id_type const &target_locality = naming::invalid_id)
Copy given component to the specified target locality.

The function copy will create a copy of the component referenced by the client side object to_copy on the locality specified with target_locality. It returns a new client side object future referring to the newly created component instance.

Return A future representing the global id of the newly (copied) component instance.

Note If the second argument is omitted (or is invalid_id) the new component instance is created on the locality of the component instance which is to be copied.

Parameters
• to_copy: [in] The client side object representing the component to copy
• target_locality: [in, optional] The locality where the copy should be created (default is same locality as source).

Template Parameters
• The: only template argument specifies the component type to create.

template <typename Component, typename DistPolicy>
future<naming::id_type> migrate (naming::id_type const &to_migrate, DistPolicy const &policy)
Migrate the given component to the specified target locality

The function migrate<Component> will migrate the component referenced by to_migrate to the locality specified with target_locality. It returns a future referring to the migrated component instance.

Return A future representing the global id of the migrated component instance. This should be the same as migrate_to.

Parameters
• to_migrate: [in] The client side representation of the component to migrate.
• policy: [in] A distribution policy which will be used to determine the locality to migrate this object to.

Template Parameters
• Component: Specifies the component type of the component to migrate.
• DistPolicy: Specifies the distribution policy to use to determine the destination locality.

template <typename Derived, typename Stub, typename DistPolicy>
Derived migrate (client_base<Derived, Stub> const &to_migrate, DistPolicy const &policy)
Migrate the given component to the specified target locality

The function migrate<Component> will migrate the component referenced by to_migrate to the locality specified with target_locality. It returns a future referring to the migrated component instance.
Return A future representing the global id of the migrated component instance. This should be the same as `migrate_to`.

Parameters
- `to_migrate`: [in] The client side representation of the component to migrate.
- `policy`: [in] A distribution policy which will be used to determine the locality to migrate this object to.

Template Parameters
- `Derived`: Specifies the component type of the component to migrate.
- `DistPolicy`: Specifies the distribution policy to use to determine the destination locality.

```
template <typename Component>
future<naming::id_type> migrate (naming::id_type const &to_migrate, naming::id_type const &target_locality)
```

Migrate the component with the given id to the specified target locality

The function `migrate<Component>` will migrate the component referenced by `to_migrate` to the locality specified with `target_locality`. It returns a future referring to the migrated component instance.

Return A future representing the global id of the migrated component instance. This should be the same as `migrate_to`.

Parameters
- `to_migrate`: [in] The global id of the component to migrate.
- `target_locality`: [in] The locality where the component should be migrated to.

Template Parameters
- `Component`: Specifies the component type of the component to migrate.

```
template <typename Derived, typename Stub>
Derived migrate (client_base<Derived, Stub> const &to_migrate, naming::id_type const &target_locality)
```

Migrate the given component to the specified target locality

The function `migrate<Component>` will migrate the component referenced by `to_migrate` to the locality specified with `target_locality`. It returns a future referring to the migrated component instance.

Return A client side representation of representing of the migrated component instance. This should be the same as `migrate_to`.

Parameters
- `to_migrate`: [in] The client side representation of the component to migrate.
- `target_locality`: [in] The id of the locality to migrate this object to.

Template Parameters
- `Derived`: Specifies the component type of the component to migrate.

Variables

```
“/runtime{locality/total}/count/component” ]
```
**binpacking_distribution_policy** `const binpacked`

A predefined instance of the binpacking `distribution_policy`. It will represent the local locality and will place all items to create here.

**colocating_distribution_policy** `const colocated`

A predefined instance of the co-locating `distribution_policy`. It will represent the local locality and will place all items to create here.

**default_distribution_policy** `const default_layout = {}`

A predefined instance of the default `distribution_policy`. It will represent the local locality and will place all items to create here.

### namespace naming

#### Functions

`id_type unmanaged(id_type const &id)`

The helper function `hpx::unmanaged` can be used to generate a global identifier which does not participate in the automatic garbage collection.

**Return** This function returns a new global id referencing the same object as the parameter `id`. The only difference is that the returned global identifier does not participate in the automatic garbage collection.

**Note** This function allows to apply certain optimizations to the process of memory management in HPX. It however requires the user to take full responsibility for keeping the referenced objects alive long enough.

**Parameters**

- `id`: [in] The id to generate the unmanaged global id from. This parameter can be itself a managed or a unmanaged global id.

### namespace parallel

### namespace v2

#### Functions

`template<typename ExPolicy, typename F>
util::detail::algorithm_result<ExPolicy>::type define_task_block (ExPolicy &&policy, F &&f)`

Constructs a task block, `tr`, using the given execution policy `policy`, and invokes the expression `f(tr)` on the user-provided object, `f`.

**Postcondition:** All tasks spawned from `f` have finished execution. A call to `define_task_block` may return on a different thread than that on which it was called.

**Template Parameters**

- `ExPolicy`: The type of the execution policy to use (deduced). It describes the manner in which the execution of the task block may be parallelized.

- `F`: The type of the user defined function to invoke inside the `define_task_block` (deduced). `F` shall be `MoveConstructible`.

**Parameters**

- `policy`: The execution policy to use for the scheduling of the iterations.

- `f`: The user defined function to invoke inside the task block. Given an lvalue `tr` of type `task_block`, the expression, `(void)f(tr)`, shall be well-formed.
Note It is expected (but not mandated) that f will (directly or indirectly) call tr.run(*callable_object*).

Exceptions

- An: *exception_list*, as specified in Exception Handling.

```cpp
template <typename F>
void define_task_block (F &&f)

Constructs a task_block, tr, and invokes the expression f(tr) on the user-provided object, f. This version uses `parallel_policy` for task scheduling.

Postcondition: All tasks spawned from f have finished execution. A call to define_task_block may return on a different thread than that on which it was called.

Template Parameters

- F: The type of the user defined function to invoke inside the define_task_block (deduced). F shall be MoveConstructible.

Parameters

- f: The user defined function to invoke inside the task block. Given an lvalue tr of type task_block, the expression, (void)f(tr), shall be well-formed.

Note It is expected (but not mandated) that f will (directly or indirectly) call tr.run(*callable_object*).

Exceptions

- An: *exception_list*, as specified in Exception Handling.

```cpp
template <typename ExPolicy, typename F>
util::detail::algorithm_result<ExPolicy>::type define_task_block_restore_thread (ExPolicy &&policy, F &&f)

Constructs a task_block, tr, and invokes the expression f(tr) on the user-provided object, f.

Postcondition: All tasks spawned from f have finished execution. A call to define_task_block_restore_thread always returns on the same thread as that on which it was called.

Template Parameters

- ExPolicy: The type of the execution policy to use (deduced). It describes the manner in which the execution of the task block may be parallelized.
- F: The type of the user defined function to invoke inside the define_task_block (deduced). F shall be MoveConstructible.

Parameters

- policy: The execution policy to use for the scheduling of the iterations.
- f: The user defined function to invoke inside the define_task_block. Given an lvalue tr of type task_block, the expression, (void)f(tr), shall be well-formed.

Exceptions

- An: *exception_list*, as specified in Exception Handling.

Note It is expected (but not mandated) that f will (directly or indirectly) call tr.run(*callable_object*).
void define_task_block_restore_thread (F && f)
Constructs a task_block, tr, and invokes the expression f(tr) on the user-provided object, f. This version
uses parallel_policy for task scheduling.

Postcondition: All tasks spawned from f have finished execution. A call to define_task_block_restore_thread always returns on the same thread as that on which it was called.

Template Parameters
• F: The type of the user defined function to invoke inside the define_task_block (deduced). F shall be MoveConstructible.

Parameters
• f: The user defined function to invoke inside the define_task_block. Given an lvalue tr of type task_block, the expression, (void)f(tr), shall be well-formed.

Exceptions
• An: exception_list, as specified in Exception Handling.

Note It is expected (but not mandated) that f will (directly or indirectly) call tr.run(callable_object).

namespace performance_counters

Functions

counter_status install_counter_type (std::string const & name,
                                        hpx::util::function_nonser<
                                            std::int64_t>
                                            & counter_value,
                                        std::string const & helptext = "/",
                                        std::string const &uom = "",
                                        error_code & ec = throws) Install a new generic performance counter type in a way, which will uninstall it automatically during shutdown.

The function install_counter_type will register a new generic counter type based on the provided function. The counter type will be automatically unregistered during system shutdown. Any consumer querying any instance of this this counter type will cause the provided function to be called and the returned value to be exposed as the counter value.

The counter type is registered such that there can be one counter instance per locality. The expected naming scheme for the counter instances is: '/objectname{locality#<*>/total}/countername' where '<*>' is a zero based integer identifying the locality the counter is created on.

Note As long as ec is not pre-initialized to hpx::throws this function doesn’t throw but returns the result code using the parameter ec. Otherwise it throws an instance of hpx::exception.

Return If successful, this function returns status_valid_data, otherwise it will either throw an exception or return an error_code from the enum counter_status (also, see note related to parameter ec).

Note The counter type registry is a locality based service. You will have to register each counter type on every locality where a corresponding performance counter will be created.

Parameters
• name: [in] The global virtual name of the counter type. This name is expected to have the format /objectname/countername.

• counter_value: [in] The function to call whenever the counter value is requested by a consumer.

• helptext: [in, optional] A longer descriptive text shown to the user to explain the nature of the counters created from this type.
counter_status install_counter_type (std::string const &name,
    hpx::util::function_nonser<
        std::vector<std::int64_t>> const &counter_value,
    std::string const &helptext = "", std::string const &uom = "/
        error_code &ec = throws) Install a new generic performance
        counter type returning an array of values in a way, that
        will uninstall it automatically during shutdown.

The function install_counter_type will register a new generic
        counter type that returns an array of values based on the
        provided function. The counter type will be automatically
        unregistered during system shutdown. Any consumer querying
        any instance of this this counter type will cause the provided
        function to be called and the returned array value to be exposed
        as the counter value.

The counter type is registered such that there can be one counter
        instance per locality. The expected naming scheme for the
        counter instances is: '/objectname{locality#<*>/total}/countername'
        where '<*>' is a zero based integer identifying the locality
        the counter is created on.

Note As long as ec is not pre-initialized to hpx::throws this function
        doesn’t throw but returns the result code using the parameter
        ec. Otherwise it throws an instance of hpx::exception.

Return If successful, this function returns status_valid_data, otherwise
        it will either throw an exception or return an error_code from
        the enum counter_status (also, see note related to parameter
        ec).

Note The counter type registry is a locality based service. You will have
        to register each counter type on every locality where a correspon-
        ding performance counter will be created.

Parameters

• name: [in] The global virtual name of the counter type. This name is expected
          to have the format /objectname/countername.

• counter_value: [in] The function to call whenever the counter value (array
          of values) is requested by a consumer.

• helptext: [in, optional] A longer descriptive text shown to the user to
          explain the nature of the counters created from this type.

• uom: [in] The unit of measure for the new performance counter type.

• ec: [in,out] this represents the error status on exit, if this is pre-initialized to
          hpx::throws the function will throw on error instead.

void install_counter_type (std::string const &name, counter_type_type,
    error_code &ec = throws) Install a new performance counter
        type in a way, which will uninstall it automatically during
        shutdown.

The function install_counter_type will register a new counter type
        based on the provided counter_type_info. The counter type will be
        automatically unregistered during system shutdown.

Return If successful, this function returns status_valid_data, otherwise it
        will either throw an exception or return an error_code from the enum
        counter_status (also, see note related to parameter ec).

Note The counter type registry is a locality based service. You will have to
        register each counter type on every locality where a corresponding
        performance counter will be created.

Note As long as ec is not pre-initialized to hpx::throws this function
        doesn’t throw but returns the result code using the parameter ec.
        Otherwise it throws an instance of hpx::exception.
Parameters

- **name**: [in] The global virtual name of the counter type. This name is expected to have the format /objectname/countername.
- **type**: [in] The type of the counters of this counter_type.
- **ec**: [in,out] this represents the error status on exit, if this is pre-initialized to hpx::throws the function will throw on error instead.

```cpp
counter_status install_counter_type (std::string const &name, counter_type type, std::string const &helptext, std::string const &uom = "", std::uint32_t version = HPX_PERFORMANCE_COUNTER_V1, error_code &ec = throws)
```

Install a new performance counter type in a way, which will uninstall it automatically during shutdown.

The function `install_counter_type` will register a new counter type based on the provided `counter_type_info`. The counter type will be automatically unregistered during system shutdown.

**Return** If successful, this function returns `status_valid_data`, otherwise it will either throw an exception or return an `error_code` from the enum `counter_status` (also, see note related to parameter `ec`).

**Note** The counter type registry is a locality based service. You will have to register each counter type on every locality where a corresponding performance counter will be created.

**Note** As long as `ec` is not pre-initialized to hpx::throws this function doesn’t throw but returns the result code using the parameter `ec`. Otherwise it throws an instance of hpx::exception.

Parameters

- **name**: [in] The global virtual name of the counter type. This name is expected to have the format /objectname/countername.
- **type**: [in] The type of the counters of this counter_type.
- **helptext**: [in] A longer descriptive text shown to the user to explain the nature of the counters created from this type.
- **uom**: [in] The unit of measure for the new performance counter type.
- **version**: [in] The version of the counter type. This is currently expected to be set to HPX_PERFORMANCE_COUNTER_V1.
- **ec**: [in,out] this represents the error status on exit, if this is pre-initialized to hpx::throws the function will throw on error instead.

```cpp
counter_status install_counter_type (std::string const &name, counter_type type, std::string const &helptext, create_counter_func const &create_counter, discover_counters_func const &discover_counters, std::uint32_t version = HPX_PERFORMANCE_COUNTER_V1, std::string const &uom = "", error_code &ec = throws)
```

Install a new generic performance counter type in a way, which will uninstall it automatically during shutdown.

The function `install_counter_type` will register a new generic counter type based on the provided `counter_type_info`. The counter type will be automatically unregistered during system shutdown.

**Note** As long as `ec` is not pre-initialized to hpx::throws this function doesn’t throw but returns the result code using the parameter `ec`. Otherwise it throws an instance of hpx::exception.
Return If successful, this function returns `status_valid_data`, otherwise it will either throw an exception or return an error_code from the enum `counter_status` (also, see note related to parameter `ec`).

Note The counter type registry is a locality based service. You will have to register each counter type on every locality where a corresponding performance counter will be created.

Parameters

- **name**: [in] The global virtual name of the counter type. This name is expected to have the format `/objectname/countername`.
- **type**: [in] The type of the counters of this counter_type.
- **helptext**: [in] A longer descriptive text shown to the user to explain the nature of the counters created from this type.
- **version**: [in] The version of the counter type. This is currently expected to be set to `HPX_PERFORMANCE_COUNTER_V1`.
- **create_counter**: [in] The function which will be called to create a new instance of this counter type.
- **discover_counters**: [in] The function will be called to discover counter instances which can be created.
- **uom**: [in] The unit of measure of the counter type (default: “”)
- **ec**: [in,out] this represents the error status on exit, if this is pre-initialized to `hpx::throws` the function will throw on error instead.

```cpp
namespace this_thread
{

Functions

```cpp
threads::thread_state_ex_enum suspend (threads::thread_state_enum state, threads::thread_id_type const &id, util::thread_description const &description = util::thread_description("this_thread::suspend"), error_code &ec = throws);
```

The function `suspend` will return control to the thread manager (suspends the current thread). It sets the new state of this thread to the thread state passed as the parameter.

Note Must be called from within a HPX-thread.

Exceptions

- If: &ec != &throws, never throws, but will set ec to an appropriate value when an error occurs. Otherwise, this function will throw an `hpx::exception` with an error code of `hpx::yield_aborted` if it is signaled with `wait_aborted`. If called outside of a HPX-thread, this function will throw an `hpx::exception` with an error code of `hpx::null_thread_id`. If this function is called while the thread-manager is not running, it will throw an `hpx::exception` with an error code of `hpx::invalid_status`.

```cpp
threads::thread_state_ex_enum suspend (threads::thread_state_enum state = threads::pending, util::thread_description const &description = util::thread_description("this_thread::suspend"), error_code &ec = throws);
```

The function `suspend` will return control to the thread manager (suspends the current thread). It sets the new state of this thread to the thread state passed as the parameter.

Note Must be called from within a HPX-thread.
Exceptions

- If \( \&ec \neq \&throws \), never throws, but will set \( ec \) to an appropriate value when an error occurs. Otherwise, this function will throw an \( hpx::exception \) with an error code of \( hpx::yield_aborted \) if it is signaled with \( wait_aborted \). If called outside of a HPX-thread, this function will throw an \( hpx::exception \) with an error code of \( hpx::null_thread_id \). If this function is called while the thread-manager is not running, it will throw an \( hpx::exception \) with an error code of \( hpx::invalid_status \).

```cpp
threads::thread_state_ex_enum suspend (util::steady_time_point const &abs_time,
    threads::thread_id_type const &id,
    util::thread_description const &description = util::thread_description("this_thread::suspend"),
    error_code &ec = throws)
```

The function \( suspend \) will return control to the thread manager (suspends the current thread). It sets the new state of this thread to \( suspended \) and schedules a wakeup for this thread at the given time.

**Note** Must be called from within a HPX-thread.

Exceptions

- If \( \&ec \neq \&throws \), never throws, but will set \( ec \) to an appropriate value when an error occurs. Otherwise, this function will throw an \( hpx::exception \) with an error code of \( hpx::yield_aborted \) if it is signaled with \( wait_aborted \). If called outside of a HPX-thread, this function will throw an \( hpx::exception \) with an error code of \( hpx::null_thread_id \). If this function is called while the thread-manager is not running, it will throw an \( hpx::exception \) with an error code of \( hpx::invalid_status \).

```cpp
threads::thread_state_ex_enum suspend (util::steady_time_point const &abs_time,
    util::thread_description const &description = util::thread_description("this_thread::suspend"),
    error_code &ec = throws)
```

The function \( suspend \) will return control to the thread manager (suspends the current thread). It sets the new state of this thread to \( suspended \) and schedules a wakeup for this thread at the given time.

**Note** Must be called from within a HPX-thread.

Exceptions

- If \( \&ec \neq \&throws \), never throws, but will set \( ec \) to an appropriate value when an error occurs. Otherwise, this function will throw an \( hpx::exception \) with an error code of \( hpx::yield_aborted \) if it is signaled with \( wait_aborted \). If called outside of a HPX-thread, this function will throw an \( hpx::exception \) with an error code of \( hpx::null_thread_id \). If this function is called while the thread-manager is not running, it will throw an \( hpx::exception \) with an error code of \( hpx::invalid_status \).

```cpp
threads::thread_state_ex_enum suspend (util::steady_duration const &rel_time,
    util::thread_description const &description = util::thread_description("this_thread::suspend"),
    error_code &ec = throws)
```

The function \( suspend \) will return control to the thread manager (suspends the current thread). It sets the new state of this thread to \( suspended \) and schedules a wakeup for this thread after the given duration.

**Note** Must be called from within a HPX-thread.

Exceptions

2.9. API reference 333
• If \&ec \(!= \&\text{throws}\), never throws, but will set \text{ec} to an appropriate value when an error occurs. Otherwise, this function will throw an \text{hpx::exception} with an error code of \text{hpx::yield_aborted} if it is signaled with \text{wait_aborted}. If called outside of a HPX-thread, this function will throw an \text{hpx::exception} with an error code of \text{hpx::null_thread_id}. If this function is called while the thread-manager is not running, it will throw an \text{hpx::exception} with an error code of \text{hpx::invalid_status}.

threads::thread_state_ex_enum \text{suspend} (util::steady_duration \text{const} &rel\_time, threads::thread_id_type \text{const} &id, util::thread_description \text{const} &\text{description} = util::thread_description("this_thread::suspend"), error_code \&\text{ec} = \text{throws})

The function \text{suspend} will return control to the thread manager (suspends the current thread). It sets the new state of this thread to \text{suspended} and schedules a wakeup for this threads after the given duration.

\textbf{Note} Must be called from within a HPX-thread.

\textbf{Exceptions}

• If \&ec \(!= \&\text{throws}\), never throws, but will set \text{ec} to an appropriate value when an error occurs. Otherwise, this function will throw an \text{hpx::exception} with an error code of \text{hpx::yield_aborted} if it is signaled with \text{wait_aborted}. If called outside of a HPX-thread, this function will throw an \text{hpx::exception} with an error code of \text{hpx::null_thread_id}. If this function is called while the thread-manager is not running, it will throw an \text{hpx::exception} with an error code of \text{hpx::invalid_status}.

threads::thread_state_ex_enum \text{suspend} (std::uint64_t \text{ms}, util::thread_description \text{const} &\text{description} = util::thread_description("this_thread::suspend"), error_code \&\text{ec} = \text{throws})

The function \text{suspend} will return control to the thread manager (suspends the current thread). It sets the new state of this thread to \text{suspended} and schedules a wakeup for this threads after the given time (specified in milliseconds).

\textbf{Note} Must be called from within a HPX-thread.

\textbf{Exceptions}

• If \&ec \(!= \&\text{throws}\), never throws, but will set \text{ec} to an appropriate value when an error occurs. Otherwise, this function will throw an \text{hpx::exception} with an error code of \text{hpx::yield_aborted} if it is signaled with \text{wait_aborted}. If called outside of a HPX-thread, this function will throw an \text{hpx::exception} with an error code of \text{hpx::null_thread_id}. If this function is called while the thread-manager is not running, it will throw an \text{hpx::exception} with an error code of \text{hpx::invalid_status}.

threads::executors::current_executor \text{get_executor} (error_code \&\text{ec} = \text{throws})

Returns a reference to the executor which was used to create the current thread.

\textbf{Exceptions}

• If \&ec \(!= \&\text{throws}\), never throws, but will set \text{ec} to an appropriate value when an error occurs. Otherwise, this function will throw an \text{hpx::exception} with an error code of \text{hpx::yield_aborted} if it is signaled with \text{wait_aborted}. If called outside of a HPX-thread, this function will throw an \text{hpx::exception} with an error code of \text{hpx::null_thread_id}. If this function is called while the thread-manager is not running, it will throw an \text{hpx::exception} with an error code of \text{hpx::invalid_status}.
threads::thread_pool_base *get_pool (error_code &ec = throws)
Returns a pointer to the pool that was used to run the current thread

Exceptions

• If: &ec != &throws, never throws, but will set ec to an appropriate value when an error occurs. Otherwise, this function will throw an hpx::exception with an error code of hpx::yield_aborted if it is signaled with wait_aborted. If called outside of a HPX-thread, this function will throw an hpx::exception with an error code of hpx::null_thread_id. If this function is called while the thread-manager is not running, it will throw an hpx::exception with an error code of hpx::invalid_status.

namespace threads

Functions

thread_self &get_self ()
The function get_self returns a reference to the (OS thread specific) self reference to the current HPX thread.

thread_self *get_self_ptr ()
The function get_self_ptr returns a pointer to the (OS thread specific) self reference to the current HPX thread.

thread_self_impl_type *get_ctx_ptr ()
The function get_ctx_ptr returns a pointer to the internal data associated with each coroutine.

thread_self *get_self_ptr_checked (error_code &ec = throws)
The function get_self_ptr_checked returns a pointer to the (OS thread specific) self reference to the current HPX thread.

thread_id_type get_self_id ()
The function get_self_id returns the HPX thread id of the current thread (or zero if the current thread is not a HPX thread).

thread_data *get_self_id_data ()
The function get_self_id_data returns the data of the HPX thread id associated with the current thread (or nullptr if the current thread is not a HPX thread).

thread_id_type get_parent_id ()
The function get_parent_id returns the HPX thread id of the current thread’s parent (or zero if the current thread is not a HPX thread).

Note This function will return a meaningful value only if the code was compiled with HPX_HAVE_THREAD_PARENT_REFERENCE being defined.

std::size_t get_parent_phase ()
The function get_parent_phase returns the HPX phase of the current thread’s parent (or zero if the current thread is not a HPX thread).

Note This function will return a meaningful value only if the code was compiled with HPX_HAVE_THREAD_PARENT_REFERENCE being defined.

std::size_t get_self_stacksize ()
The function get_self_stacksize returns the stack size of the current thread (or zero if the current thread is not a HPX thread).
The function `get_parent_locality_id()` returns the id of the locality of the current thread’s parent (or zero if the current thread is not a HPX thread).

**Note** This function will return a meaningful value only if the code was compiled with `HPX_HAVE_THREAD_PARENT_REFERENCE` being defined.

The function `get_self_component_id()` returns the lva of the component the current thread is acting on.

**Note** This function will return a meaningful value only if the code was compiled with `HPX_HAVE_THREAD_TARGET_ADDRESS` being defined.

The function `get_thread_count()` returns the number of currently known threads.

**Note** If state == `unknown` this function will not only return the number of currently existing threads, but will add the number of registered task descriptions (which have not been converted into threads yet).

**Parameters**

- `state`: [in] This specifies the thread-state for which the number of threads should be retrieved.

The function `get_thread_count()` returns the number of currently known threads.

**Note** If state == `unknown` this function will not only return the number of currently existing threads, but will add the number of registered task descriptions (which have not been converted into threads yet).

**Parameters**

- `priority`: [in] This specifies the thread-priority for which the number of threads should be retrieved.
- `state`: [in] This specifies the thread-state for which the number of threads should be retrieved.

The function `enumerate_threads()` will invoke the given function `f` for each thread with a matching thread state.

**Parameters**

- `f`: [in] The function which should be called for each matching thread. Returning ‘false’ from this function will stop the enumeration process.
- `state`: [in] This specifies the thread-state for which the threads should be enumerated.

Set the thread state of the thread referenced by the thread_id `id`.

**Note** If the thread referenced by the parameter `id` is in `thread_state::active` state this function schedules a new thread which will set the state of the thread as soon as its not active anymore. The function returns `thread_state::active` in this case.
Return This function returns the previous state of the thread referenced by the id parameter. It will return one of the values as defined by the thread_state enumeration. If the thread is not known to the thread-manager the return value will be thread_state::unknown.

Note As long as ec is not pre-initialized to hpx::throws this function doesn’t throw but returns the result code using the parameter ec. Otherwise it throws an instance of hpx::exception.

Parameters

- id: [in] The thread id of the thread the state should be modified for.
- state: [in] The new state to be set for the thread referenced by the id parameter.
- stateex: [in] The new extended state to be set for the thread referenced by the id parameter.
- priority:
- ec: [in,out] this represents the error status on exit, if this is pre-initialized to hpx::throws the function will throw on error instead.

```cpp
thread_id_type set_thread_state(thread_id_type const &id, util::steady_time_point const &abs_time, std::atomic<bool> *started, thread_state_enum state = pending, thread_state_ex_enum stateex = wait_timeout, thread_priority priority = thread_priority_normal, bool retry_on_active = true, error_code &ec = throws)
```

Set the thread state of the thread referenced by the thread_id id.

Set a timer to set the state of the given thread to the given new value after it expired (at the given time)

Return

Note As long as ec is not pre-initialized to hpx::throws this function doesn’t throw but returns the result code using the parameter ec. Otherwise it throws an instance of hpx::exception.

Parameters

- id: [in] The thread id of the thread the state should be modified for.
- abs_time: [in] Absolute point in time for the new thread to be run
- started: [in,out] A helper variable allowing to track the state of the timer helper thread
- state: [in] The new state to be set for the thread referenced by the id parameter.
- stateex: [in] The new extended state to be set for the thread referenced by the id parameter.
- priority:
- ec: [in,out] this represents the error status on exit, if this is pre-initialize to hpx::throws the function will throw on error instead.

```cpp
thread_id_type set_thread_state(thread_id_type const &id, util::steady_time_point const &abs_time, thread_state_enum state = pending, thread_state_ex_enum stateex = wait_timeout, thread_priority priority = thread_priority_normal, bool retry_on_active = true, error_code &ec = throws)
```

```cpp
thread_id_type set_thread_state(thread_id_type const &id, util::steady_duration const &rel_time, thread_state_enum state = pending, thread_state_ex_enum stateex = wait_timeout, thread_priority priority = thread_priority_normal, bool retry_on_active = true, error_code &ec = throws)
```

Set the thread state of the thread referenced by the thread_id id.
Set a timer to set the state of the given thread to the given new value after it expired (after the given duration)

**Return**

**Note** As long as ec is not pre-initialized to hpx::throws this function doesn’t throw but returns the result code using the parameter ec. Otherwise it throws an instance of hpx::exception.

**Parameters**

- **id**: [in] The thread id of the thread the state should be modified for.
- **rel_time**: [in] Time duration after which the new thread should be run
- **state**: [in] The new state to be set for the thread referenced by the id parameter.
- **stateex**: [in] The new extended state to be set for the thread referenced by the id parameter.
- **priority**:
- **ec**: [in,out] this represents the error status on exit, if this is pre-initialized to hpx::throws the function will throw on error instead.

```cpp
util::thread_description get_thread_description(thread_id_type const &id, error_code &ec = throws)
```

The function get_thread_description is part of the thread related API allows to query the description of one of the threads known to the thread-manager.

**Return** This function returns the description of the thread referenced by the id parameter. If the thread is not known to the thread-manager the return value will be the string “<unknown>”.

**Note** As long as ec is not pre-initialized to hpx::throws this function doesn’t throw but returns the result code using the parameter ec. Otherwise it throws an instance of hpx::exception.

**Parameters**

- **id**: [in] The thread id of the thread being queried.
- **ec**: [in,out] this represents the error status on exit, if this is pre-initialized to hpx::throws the function will throw on error instead.

```cpp
util::thread_description set_thread_description(thread_id_type const &id, util::thread_description const &desc = util::thread_description(), error_code &ec = throws)
```

```cpp
util::thread_description get_thread_lco_description(thread_id_type const &id, error_code &ec = throws)
```

```cpp
util::thread_description set_thread_lco_description(thread_id_type const &id, util::thread_description const &desc = util::thread_description(), error_code &ec = throws)
```

```
thread_state get_thread_state(thread_id_type const &id, error_code &ec = throws)
```

The function get_thread_backtrace is part of the thread related API allows to query the currently stored thread back trace (which is captured during thread suspension).

**Return** This function returns the currently captured stack back trace of the thread referenced by the id parameter. If the thread is not known to the thread-manager the return value will be the zero.
Note As long as \textit{ec} is not pre-initialized to \texttt{hpx::throws} this function doesn’t throw but returns the result code using the parameter \textit{ec}. Otherwise it throws an instance of \texttt{hpx::exception}. The function get_thread_state is part of the thread related API. It queries the state of one of the threads known to the thread-manager.

Return This function returns the thread state of the thread referenced by the \textit{id} parameter. If the thread is not known to the thread-manager the return value will be \texttt{terminated}.

Note As long as \textit{ec} is not pre-initialized to \texttt{hpx::throws} this function doesn’t throw but returns the result code using the parameter \textit{ec}. Otherwise it throws an instance of \texttt{hpx::exception}.

Parameters

- \textit{id}: [in] The thread id of the thread being queried.
- \textit{ec}: [in,out] this represents the error status on exit, if this is pre-initialized to \texttt{hpx::throws} the function will throw on error instead.

Parameters

- \textit{id}: [in] The thread id of the thread the state should be modified for.
- \textit{ec}: [in,out] this represents the error status on exit, if this is pre-initialized to \texttt{hpx::throws} the function will throw on error instead.

\begin{verbatim}
std::size_t get_thread_phase (thread_id_type const &id, error_code &ec = throws)
\end{verbatim}

The function get_thread_phase is part of the thread related API. It queries the phase of one of the threads known to the thread-manager.

Return This function returns the thread phase of the thread referenced by the \textit{id} parameter. If the thread is not known to the thread-manager the return value will be \texttt{~0}.

Note As long as \textit{ec} is not pre-initialized to \texttt{hpx::throws} this function doesn’t throw but returns the result code using the parameter \textit{ec}. Otherwise it throws an instance of \texttt{hpx::exception}.

Parameters

- \textit{id}: [in] The thread id of the thread the phase should be modified for.
- \textit{ec}: [in,out] this represents the error status on exit, if this is pre-initialized to \texttt{hpx::throws} the function will throw on error instead.


\begin{verbatim}
bool get_thread_interruption_enabled (thread_id_type const &id, bool &enable, error_code &ec = throws)
\end{verbatim}

Returns whether the given thread can be interrupted at this point.

Return This function returns \texttt{true} if the given thread can be interrupted at this point in time. It will return \texttt{false} otherwise.

Note As long as \textit{ec} is not pre-initialized to \texttt{hpx::throws} this function doesn’t throw but returns the result code using the parameter \textit{ec}. Otherwise it throws an instance of \texttt{hpx::exception}.

Parameters

- \textit{id}: [in] The thread id of the thread which should be queried.
- \textit{ec}: [in,out] this represents the error status on exit, if this is pre-initialized to \texttt{hpx::throws} the function will throw on error instead.

\begin{verbatim}
bool set_thread_interruption_enabled (thread_id_type const &id, bool enable, error_code &ec = throws)
\end{verbatim}

Set whether the given thread can be interrupted at this point.
Return  This function returns the previous value of whether the given thread could have been interrupted.

Note As long as \( ec \) is not pre-initialized to \( hpx::throws \) this function doesn’t throw but returns the result code using the parameter \( ec \). Otherwise it throws an instance of \( hpx::exception \).

Parameters

- \( \text{id} \): [in] The thread id of the thread which should receive the new value.
- \( \text{enable} \): [in] This value will determine the new interruption enabled status for the given thread.
- \( \text{ec} \): [in,out] this represents the error status on exit, if this is pre-initialized to \( hpx::throws \) the function will throw on error instead.

bool \text{get_thread_interruption_requested} (\text{thread_id_type const &id, error_code &ec = throws})

Returns whether the given thread has been flagged for interruption.

Return  This function returns \text{true} if the given thread was flagged for interruption. It will return \text{false} otherwise.

Note As long as \( ec \) is not pre-initialized to \( hpx::throws \) this function doesn’t throw but returns the result code using the parameter \( ec \). Otherwise it throws an instance of \( hpx::exception \).

Parameters

- \( \text{id} \): [in] The thread id of the thread which should be queried.
- \( \text{ec} \): [in,out] this represents the error status on exit, if this is pre-initialized to \( hpx::throws \) the function will throw on error instead.

void \text{interrupt_thread} (\text{thread_id_type const &id, bool flag, error_code &ec = throws})

Flag the given thread for interruption.

Note As long as \( ec \) is not pre-initialized to \( hpx::throws \) this function doesn’t throw but returns the result code using the parameter \( ec \). Otherwise it throws an instance of \( hpx::exception \).

Parameters

- \( \text{id} \): [in] The thread id of the thread which should be interrupted.
- \( \text{flag} \): [in] The flag encodes whether the thread should be interrupted (if it is \text{true}) or ‘uninterrupted’ (if it is \text{false}).
- \( \text{ec} \): [in,out] this represents the error status on exit, if this is pre-initialized to \( hpx::throws \) the function will throw on error instead.

void \text{interrupt_thread} (\text{thread_id_type const &id, error_code &ec = throws})

void \text{interruption_point} (\text{thread_id_type const &id, error_code &ec = throws})

Interrupt the current thread at this point if it was canceled. This will throw a \text{thread_interrupted} exception, which will cancel the thread.

Note As long as \( ec \) is not pre-initialized to \( hpx::throws \) this function doesn’t throw but returns the result code using the parameter \( ec \). Otherwise it throws an instance of \( hpx::exception \).

Parameters

- \( \text{id} \): [in] The thread id of the thread which should be interrupted.
- \( \text{ec} \): [in,out] this represents the error status on exit, if this is pre-initialized to \( hpx::throws \) the function will throw on error instead.
threads::thread_priority get_thread_priority (thread_id_type const &id, error_code &ec = throws)

Return priority of the given thread

Note As long as ec is not pre-initialized to hpx::throws this function doesn’t throw but returns the result code using the parameter ec. Otherwise it throws an instance of hpx::exception.

Parameters

• id: [in] The thread id of the thread whose priority is queried.
• ec: [in,out] this represents the error status on exit, if this is pre-initialized to hpx::throws the function will throw on error instead.

std::ptrdiff_t get_stack_size (thread_id_type const &id, error_code &ec = throws)

Return stack size of the given thread

Note As long as ec is not pre-initialized to hpx::throws this function doesn’t throw but returns the result code using the parameter ec. Otherwise it throws an instance of hpx::exception.

Parameters

• id: [in] The thread id of the thread whose priority is queried.
• ec: [in,out] this represents the error status on exit, if this is pre-initialized to hpx::throws the function will throw on error instead.

threads::executors::current_executor get_executor (thread_id_type const &id, error_code &ec = throws)

Returns a reference to the executor which was used to create the given thread.

Exceptions

• If &ec != &throws, never throws, but will set ec to an appropriate value when an error occurs. Otherwise, this function will throw an hpx::exception with an error code of hpx::yield_aborted if it is signaled with wait_aborted. If called outside of a HPX-thread, this function will throw an hpx::exception with an error code of hpx::null_thread_id. If this function is called while the thread-manager is not running, it will throw an hpx::exception with an error code of hpx::invalid_status.

threads::thread_pool_base *get_pool (thread_id_type const &id, error_code &ec = throws)

Returns a pointer to the pool that was used to run the current thread

Exceptions

• If &ec != &throws, never throws, but will set ec to an appropriate value when an error occurs. Otherwise, this function will throw an hpx::exception with an error code of hpx::yield_aborted if it is signaled with wait_aborted. If called outside of a HPX-thread, this function will throw an hpx::exception with an error code of hpx::null_thread_id. If this function is called while the thread-manager is not running, it will throw an hpx::exception with an error code of hpx::invalid_status.

namespace policies
## Enums

**enum scheduler_mode**

This enumeration describes the possible modes of a scheduler.

*Values:*

- **nothing_special = 0x000**
  
  can be used to disable all other options.

  As the name suggests, this option

- **do_background_work = 0x001**

  The scheduler will periodically call a provided callback function from a special HPX thread to enable performing background-work, for instance driving networking progress or garbage-collect AGAS.

- **reduce_thread_priority = 0x002**

  The kernel priority of the os-thread driving the scheduler will be reduced below normal.

- **delay_exit = 0x004**

  The scheduler will wait for some unspecified amount of time before exiting the scheduling loop while being terminated to make sure no other work is being scheduled during processing the shutdown request.

- **fast_idle_mode = 0x008**

  Some schedulers have the capability to act as ‘embedded’ schedulers. In this case it needs to periodically invoke a provided callback into the outer scheduler more frequently than normal. This option enables this behavior.

- **enable_elasticity = 0x010**

  This option allows for the scheduler to dynamically increase and reduce the number of processing units it runs on. Setting this value not succeed for schedulers that do not support this functionality.

- **enable_stealing = 0x020**

  that support work thread/stealing to enable/disable it

  This option allows schedulers

- **enable_stealing_numa = 0x040**

  that support it to disallow stealing between numa domains

  This option allows schedulers

- **assign_work_round_robin = 0x080**

  that support it to add tasks round robin to queues on each core

  This option tells schedulers

- **assign_work_thread_parent = 0x100**

  This option tells schedulers that support it to add tasks round to the same core/queue that the parent task is running on

- **steal_high_priority_first = 0x200**

  This option tells schedulers that support it to always (try to) steal high priority tasks from other queues before finishing their own lower priority tasks

- **steal_after_local = 0x400**

  This option tells schedulers that support it to steal tasks only when their local queues are empty

- **enable_idle_backoff = 0x800**

  schedulers to explicitly disable exponential idle-back off
This option allows for certain
\[
\text{do\_background\_work | reduce\_thread\_priority | delay\_exit | enable\_stealing | enable\_stealing\_numa | assign\_work\_round\_robin | steal\_after\_local | enable\_idle\_backoff}
\]
This option represents the default mode.
\[
\text{do\_background\_work | reduce\_thread\_priority | delay\_exit | fast\_idle\_mode | enable\_elasticity | enable\_stealing | enable\_stealing\_numa | assign\_work\_round\_robin | assign\_work\_thread\_parent | steal\_high\_priority\_first | steal\_after\_local | enable\_idle\_backoff}
\]

namespace traits

namespace util

## Functions

void **attach**

Attaches a debugger if category is equal to the configuration entry hpx.attach-debugger.

`template <typename Mapper, typename... T>
<unspecified> hpx::util::map_pack(Mapper && mapper, T &&... pack)`

Maps the pack with the given mapper.

This function tries to visit all plain elements which may be wrapped in:

- homogeneous containers (`std::vector`, `std::list`)
- heterogenous containers (`hpx::tuple`, `std::pair`, `std::array`) and re-assembles the pack with the result of the mapper. Mapping from one type to a different one is supported.

Elements that aren’t accepted by the mapper are routed through and preserved through the hierarchy.

```cpp
// Maps all integers to floats
map_pack([](int value) {
    return float(value);
},
    i, hpx::util::make_tuple(2, std::vector<int>{3, 4}), 5);
```

**Return** The mapped element or in case the pack contains multiple elements, the pack is wrapped into a `hpx::tuple`.

**Exceptions**

- `std::exception`: like objects which are thrown by an invocation to the mapper.

**Parameters**

- `mapper`: A callable object, which accept an arbitrary type and maps it to another type or the same one.
- `pack`: An arbitrary variadic pack which may contain any type.

`template <typename Visitor, typename... T>
auto traverse_pack_async (Visitor &&visitor, T &&... pack)`

Traverses the pack with the given visitor in an asynchronous way.
This function works in the same way as `traverse_pack`, however, we are able to suspend and continue
the traversal at later time. Thus we require a visitor callable object which provides three `operator()`
overloads as depicted by the code sample below:

```cpp
struct my_async_visitor
{
    template <typename T>
    bool operator()(async_traverse_visit_tag, T&& element)
    {
        return true;
    }

    template <typename T, typename N>
    void operator()(async_traverse_detach_tag, T&& element, N&& next)
    {
    }

    template <typename T>
    void operator()(async_traverse_complete_tag, T&& pack)
    {
    }
};
```

See `traverse_pack` for a detailed description about the traversal behavior and capabilities.

**Return** A `hpx::intrusive_ptr` that references an instance of the given visitor object.

**Parameters**

- **visitor**: A visitor object which provides the three `operator()` overloads that were
  described above. Additionally the visitor must be compatible for referencing it from a
  `hpx::intrusive_ptr`. The visitor should must have a virtual destructor!

- **pack**: The arbitrary parameter pack which is traversed asynchronously. Nested objects inside
  containers and tuple like types are traversed recursively.

```
template <typename Allocator, typename Visitor, typename... T>
auto traverse_pack_async_allocator(Allocator const &alloc, Visitor &visitor, T&&...
pack)
Traverses the pack with the given visitor in an asynchronous way.

This function works in the same way as `traverse_pack`, however, we are able to suspend and continue
the traversal at later time. Thus we require a visitor callable object which provides three `operator()`
overloads as depicted by the code sample below:

```cpp
struct my_async_visitor
{
    template <typename T>
    bool operator()(async_traverse_visit_tag, T&& element)
    {
        return true;
    }

    template <typename T, typename N>
    void operator()(async_traverse_detach_tag, T&& element, N&& next)
    {
    }

    template <typename T>
    void operator()(async_traverse_complete_tag, T&& pack)
    {
    }
};
```
See `traverse_pack` for a detailed description about the traversal behavior and capabilities.

**Return** A `hpx::intrusive_ptr` that references an instance of the given visitor object.

**Parameters**

- **visitor**: A visitor object which provides the three `operator()` overloads that were described above. Additionally the visitor must be compatible for referencing it from a `hpx::intrusive_ptr`. The visitor should must have a virtual destructor!

- **pack**: The arbitrary parameter pack which is traversed asynchronously. Nested objects inside containers and tuple like types are traversed recursively.

- **alloc**: Allocator instance to use to create the traversal frame.

```cpp
template <typename... Args>
auto unwrap (Args&&... args)
```

A helper function for retrieving the actual result of any `hpx::lcos::future` like type which is wrapped in an arbitrary way.

Unwraps the given pack of arguments, so that any `hpx::lcos::future` object is replaced by its future result type in the argument pack:

- `hpx::future<int>` -> `int`
- `hpx::future<std::vector<float>>` -> `std::vector<float>`
- `std::vector<future<float>>` -> `std::vector<float>`

The function is capable of unwrapping `hpx::lcos::future` like objects that are wrapped inside any container or tuple like type, see `hpx::util::map_pack()` for a detailed description about which surrounding types are supported. Non `hpx::lcos::future` like types are permitted as arguments and passed through.

```cpp
// Single arguments
int i1 = hpx::util::unwrap(hpx::lcos::make_ready_future(0));
```

```cpp
// Multiple arguments
hpx::tuple<int, int> i2 =
    hpx::util::unwrap(hpx::lcos::make_ready_future(1),
                      hpx::lcos::make_ready_future(2));
```

**Note** This function unwraps the given arguments until the first traversed nested `hpx::lcos::future` which corresponds to an unwrapping depth of one. See `hpx::util::unwrap_n()` for a function which unwraps the given arguments to a particular depth or `hpx::util::unwrap_all()` that unwraps all future like objects recursively which are contained in the arguments.

**Return** Depending on the count of arguments this function returns a `hpx::util::tuple` containing the unwrapped arguments if multiple arguments are given. In case the function is called with a single argument, the argument is unwrapped and returned.

**Parameters**

- **args**: the arguments that are unwrapped which may contain any arbitrary future or non future type.

**Exceptions**
• std::exception: like objects in case any of the given wrapped hpx::lcos::future objects were resolved through an exception. See hpx::lcos::future::get() for details.

```cpp
template <std::size_t Depth, typename... Args>
auto unwrap_n (Args&&... args)
    An alterntive version of hpx::util::unwrap(), which unwraps the given arguments to a certain depth of hpx::lcos::future like objects.

See unwrap for a detailed description.

Template Parameters
• Depth: The count of hpx::lcos::future like objects which are unwrapped maximally.
```

```cpp
template <typename... Args>
auto unwrap_all (Args&&... args)
    An alternative version of hpx::util::unwrap(), which unwraps the given arguments recursively so that all contained hpx::lcos::future like objects are replaced by their actual value.

See hpx::util::unwrap() for a detailed description.
```

```cpp
template <typename T>
auto unwrapping (T &&callable)
    Returns a callable object which unwraps its arguments upon invocation using the hpx::util::unwrap() function and then passes the result to the given callable object.

auto callable = hpx::util::unwrapping([](int left, int right) {
    return left + right;
});

int i1 = callable(hpx::lcos::make_ready_future(1),
    hpx::lcos::make_ready_future(2));
```

See hpx::util::unwrap() for a detailed description.

Parameters
• callable: the callable object which is called with the result of the corresponding unwrap function.

```cpp
template <std::size_t Depth, typename T>
auto unwrapping_n (T &&&callable)
    Returns a callable object which unwraps its arguments upon invocation using the hpx::util::unwrap_n() function and then passes the result to the given callable object.

See hpx::util::unwrap() for a detailed description.
```

```cpp
template <typename T>
auto unwrapping_all (T &&&callable)
    Returns a callable object which unwraps its arguments upon invocation using the hpx::util::unwrap_all() function and then passes the result to the given callable object.

See hpx::util::unwrap() for a detailed description.
```
<hpx/parallel/executors/execution.hpp>#include <hpx/parallel/executors/post_policy_dispatch.hpp>#include <hpx/runtime/naming/id_type.hpp>#include <hpx/components/component_storage/server/migrate_from_storage.hpp>

file migrate_to_storage.hpp
#include <hpx/config.hpp>#include <hpx/lcos/future.hpp>#include <hpx/runtime/components/client_base.hpp>#include <hpx/functional/bind_back.hpp>#include <hpx/memory/intrusive_ptr.hpp>#include <hpx/runtime/agas/interface.hpp>#include <hpx/runtime/components/component_type.hpp>#include <hpx/functional/unique_function.hpp>#include <hpx/preprocessor/cat.hpp>#include <hpx/preprocessor/expand.hpp>#include <hpx/preprocessor/nargs.hpp>#include <hpx/preprocessor/strip_parens.hpp>#include <hpx/runtime/naming_fwd.hpp>#include <hpx/thread_support/atomic_count.hpp>#include <hpx/traits/component_type_database.hpp>#include <cstdint>#include <hpx/type_support/decay.hpp>#include <string>#include <hpx/runtime/components_fwd.hpp>#include <hpx/traits/managed_component_policies.hpp>#include <cstddef>#include <hpx/runtime/launch_policy.hpp>#include <hpx/runtime/naming/name.hpp>#include <hpx/allocator_support/internal_allocator.hpp>#include <hpx/serialization/traits/is_bitwise_serializable.hpp>#include <hpx/traits/promise_local_result.hpp>#include <functional>#include <iosfwd>#include <mutex>#include <vector>

file hpx_finalize.hpp
#include <hpx/config.hpp>#include <hpx/errors.hpp>

file hpx_init.hpp
#include <hpx/config.hpp>#include <hpx/errors.hpp>#include <hpx/hpx_finalize.hpp>#include <hpx/program_options.hpp>#include <hpx/runtime/runtime_mode.hpp>#include <hpx/runtime/shutdown_function.hpp>#include <hpx/functional/function.hpp>#include <cstring>

file hpx_start.hpp
#include <hpx/config.hpp>#include <hpx/hpx_finalize.hpp>#include <hpx/program_options.hpp>#include <hpx/runtime/runtime_mode.hpp>#include <hpx/runtime/shutdown_function.hpp>#include <hpx/functional/function.hpp>#include <cstring>

file hpx_suspend.hpp
#include <hpx/errors.hpp>

file split_future.hpp
file wait_all.hpp
file wait_any.hpp
file wait_each.hpp
file when_all.hpp
file when_any.hpp
file when_each.hpp
file when_some.hpp
file task_block.hpp

#include <hpx/config.hpp>
#include <hpx/async.hpp>
#include <hpx/async_launch_policy_dispatch.hpp>
#include <hpx/assertion.hpp>
#include <hpx/coroutines/thread_enums.hpp>
#include <hpx/functional/deferred_call.hpp>
#include <hpx/functional/invoke.hpp>
#include <hpx/functional/traits/is_action.hpp>
#include <hpx/lcos/async_fwd.hpp>
#include <hpx/lcos/future.hpp>
#include <hpx/lcos/local/futures_factory.hpp>
#include <hpx/allocator_support/allocator_deleter.hpp>
#include <hpx/allocator_support/internal_allocator.hpp>
#include <hpx/errors.hpp>
#include <hpx/lcos/detail/future_data.hpp>
#include <hpx/memory/intrusive_ptr.hpp>
#include <hpx/runtime/get_worker_thread_num.hpp>
#include <cstring>
#include <hpx/runtime/launch_policy.hpp>
#include <hpx/runtime/threads/thread_data_fwd.hpp>
#include <iostream>
#include <exception>
#include <memory>
#include <type_traits>
#include <cstdint>
#include <memory>
#include <type_traits>
#include <utility>
#include <hpx/lcos/future_fwd.hpp>
#include <hpx/runtime/applier/detail/apply_implementations_fwd.hpp>
#include <hpx/lcos_fwd.hpp>


<file>
#include <hpx/config.hpp>
#include <hpx/errors.hpp>
#include <hpx/functional/function.hpp>
#include <cstddef>
#include <cstdint>
#include <string>
#include <vector>

Defines

HPX_REGISTER_ACTION_DECLARATION(...)  
Declare the necessary component action boilerplate code.

The macro HPX_REGISTER_ACTION_DECLARATION can be used to declare all the boilerplate code which is required for proper functioning of component actions in the context of HPX.

The parameter action is the type of the action to declare the boilerplate for.

This macro can be invoked with an optional second parameter. This parameter specifies a unique name of the action to be used for serialization purposes. The second parameter has to be specified if the first parameter is not usable as a plain (non-qualified) C++ identifier, i.e. the first parameter contains special characters which cannot be part of a C++ identifier, such as ‘<’, ‘>’, or ‘:’.

2.9. API reference 351
namespace app
{
    // Define a simple component exposing one action 'print_greeting'
    class HPX_COMPONENT_EXPORT server
    {
        public hpx::components::simple_component_base<server>
        {
            void print_greeting()
            {
                hpx::cout << "Hey, how are you?\n" << hpx::flush;
            }

            // Component actions need to be declared, this also defines the
            // type 'print_greeting_action' representing the action.
            HPX_DEFINE_COMPONENT_ACTION(server,
                print_greeting, print_greeting_action);
        }
    }
}

// Declare boilerplate code required for each of the component actions.
HPX_REGISTER_ACTION_DECLARATION(app::server::print_greeting_action);

Example:

Note This macro has to be used once for each of the component actions defined using one of the
HPX_DEFINE_COMPONENT_ACTION macros. It has to be visible in all translation units using
the action, thus it is recommended to place it into the header file defining the component.

HPX_REGISTER_ACTION_DECLARATION_(...)

HPX_REGISTER_ACTION_DECLARATION_1 (action)

HPX_REGISTER_ACTION (...)
Define the necessary component action boilerplate code.

The macro HPX_REGISTER_ACTION can be used to define all the boilerplate code which is required for
proper functioning of component actions in the context of HPX.

The parameter action is the type of the action to define the boilerplate for.

This macro can be invoked with an optional second parameter. This parameter specifies a unique name
of the action to be used for serialization purposes. The second parameter has to be specified if the first
parameter is not usable as a plain (non-qualified) C++ identifier, i.e. the first parameter contains special
characters which cannot be part of a C++ identifier, such as ‘<’, ‘>’, or ‘:’.

Note This macro has to be used once for each of the component actions defined using one of the
HPX_DEFINE_COMPONENT_ACTION or HPX_DEFINE_PLAIN_ACTION macros. It has to oc-
cur exactly once for each of the actions, thus it is recommended to place it into the source file defining
the component.

Note Only one of the forms of this macro HPX_REGISTER_ACTION or HPX_REGISTER_ACTION_ID
should be used for a particular action, never both.

HPX_REGISTER_ACTION_ID (action, actionname, actionid)
Define the necessary component action boilerplate code and assign a predefined unique id to the action.

The macro HPX_REGISTER_ACTION can be used to define all the boilerplate code which is required for
proper functioning of component actions in the context of HPX.
The parameter \textit{action} is the type of the action to define the boilerplate for.

The parameter \textit{actionname} specifies an unique name of the action to be used for serialization purposes. The second parameter has to be usable as a plain (non-qualified) C++ identifier, it should not contain special characters which cannot be part of a C++ identifier, such as ‘<’, ‘>’, or ‘:’.

The parameter \textit{actionid} specifies an unique integer value which will be used to represent the action during serialization.

\textbf{Note} This macro has to be used once for each of the component actions defined using one of the \textit{HPX_DEFINE_COMPONENT_ACTION} or global actions \textit{HPX_DEFINE_PLAIN_ACTION} macros. It has to occur exactly once for each of the actions, thus it is recommended to place it into the source file defining the component.

\textbf{Note} Only one of the forms of this macro \textit{HPX_REGISTER_ACTION} or \textit{HPX_REGISTER_ACTION_ID} should be used for a particular action, never both.

\textit{file} \texttt{component\_action.hpp}

\begin{verbatim}
#include <hpx/config.hpp>
#include <hpx/preprocessor/cat.hpp>
#include <hpx/preprocessor/expand.hpp>
#include <hpx/preprocessor/nargs.hpp>
#include <hpx/runtime/actions/basic_action.hpp>
#include <hpx/runtime/components/pinned_ptr.hpp>
#include <hpx/runtime/naming/address.hpp>
#include <hpx/traits/is_client.hpp>
#include <hpx/traits/is_future.hpp>
#include <boost/utility/string_ref.hpp>
#include <cstdlib>
#include <sstream>
#include <string>
#include <type_traits>
#include <utility>
#include <hpx/config/warnings_prefix.hpp>
#include <hpx/config/warnings_suffix.hpp>

Defines

\textbf{HPX_DEFINE_COMPONENT_ACTION} (...) Registers a member function of a component as an action type with HPX.

The macro \textbf{HPX_DEFINE_COMPONENT_ACTION} can be used to register a member function of a component as an action type named \textit{action\_type}.

The parameter \textit{component} is the type of the component exposing the member function \textit{func} which should be associated with the newly defined action type. The parameter \textit{action\_type} is the name of the action type to register with HPX.

\begin{verbatim}
namespace app
{
    // Define a simple component exposing one action 'print\_greeting'
    class HPX_COMPONENT_EXPORT server
    {
        public

        void print\_greeting() const
        {
            hpx\:cout \text{\textless} \text{"Hey, how are you?\n\textless} \text{hpx\:flush;
        }

        // Component actions need to be declared, this also defines the
        // type 'print\_greeting\_action' representing the action.
        HPX_DEFINE_COMPONENT_ACTION(server, print\_greeting,
                                     print\_greeting\_action);
    }
}

Example:

The first argument must provide the type name of the component the action is defined for.

\end{verbatim}
\end{verbatim}
The second argument must provide the member function name the action should wrap.

The default value for the third argument (the typename of the defined action) is derived from the name of the function (as passed as the second argument) by appending ‘_action’. The third argument can be omitted only if the second argument with an appended suffix ‘_action’ resolves to a valid, unqualified C++ type name.

**Note** The macro `HPX_DEFINE_COMPONENT_ACTION` can be used with 2 or 3 arguments. The third argument is optional.

```cpp
#define HPX_DEFINE_PLAIN_ACTION(...) Defines a plain action type.
```

```cpp
namespace app {
    void some_global_function(double d) {
        cout << d;
    }

    // This will define the action type 'app::some_global_action' which
    // represents the function 'app::some_global_function'.
    HPX_DEFINE_PLAIN_ACTION(some_global_function, some_global_action);
}
```

**Example:**

**Note** Usually this macro will not be used in user code unless the intent is to avoid defining the action_type in global namespace. Normally, the use of the macro `HPX_PLAIN_ACTION` is recommended.

**Note** The macro `HPX_DEFINE_PLAIN_ACTION` can be used with 1 or 2 arguments. The second argument is optional. The default value for the second argument (the typename of the defined action) is derived from the name of the function (as passed as the first argument) by appending ‘_action’. The second argument can be omitted only if the first argument with an appended suffix ‘_action’ resolves to a valid, unqualified C++ type name.

```cpp
#define HPX_DECLARE_PLAIN_ACTION(...) Declares a plain action type.
```

```cpp
#define HPX_PLAIN_ACTION(...) Defines a plain action type based on the given function `func` and registers it with HPX.
```

The macro `HPX_PLAIN_ACTION` can be used to define a plain action (e.g. an action encapsulating a global or free function) based on the given function `func`. It defines the action type `name` representing the given function. This macro additionally registers the newly define action type with HPX.

354 Chapter 2. What’s so special about HPX?
The parameter `func` is a global or free (non-member) function which should be encapsulated into a plain action. The parameter `name` is the name of the action type defined by this macro.

```cpp
namespace app {
    void some_global_function(double d)
    {
        cout << d;
    }
}
// This will define the action type 'some_global_action' which represents the function 'app::some_global_function'.
HPX_PLAIN_ACTION(app::some_global_function, some_global_action);
```

Example:

Note The macro `HPX_PLAIN_ACTION` has to be used at global namespace even if the wrapped function is located in some other namespace. The newly defined action type is placed into the global namespace as well.

Note The macro `HPX_PLAIN_ACTION_ID` can be used with 1, 2, or 3 arguments. The second and third arguments are optional. The default value for the second argument (the typename of the defined action) is derived from the name of the function (as passed as the first argument) by appending ‘_action’. The second argument can be omitted only if the first argument with an appended suffix ‘_action’ resolves to a valid, unqualified C++ type name. The default value for the third argument is `hpx::components::factory_check`.

Note Only one of the forms of this macro `HPX_PLAIN_ACTION` or `HPX_PLAIN_ACTION_ID` should be used for a particular action, never both.

`HPX_PLAIN_ACTION_ID` (func, name, id)
Defines a plain action type based on the given function `func` and registers it with HPX.

The macro `HPX_PLAIN_ACTION_ID` can be used to define a plain action (e.g. an action encapsulating a global or free function) based on the given function `func`. It defines the action type `actionname` representing the given function. The parameter `actionid` specifies an unique integer value which will be used to represent the action during serialization.

The parameter `func` is a global or free (non-member) function which should be encapsulated into a plain action. The parameter `name` is the name of the action type defined by this macro.

The second parameter has to be usable as a plain (non-qualified) C++ identifier, it should not contain special characters which cannot be part of a C++ identifier, such as ‘<’, ‘>’, or ‘;’.

```cpp
namespace app {
    void some_global_function(double d)
    {
        cout << d;
    }
}
// This will define the action type 'some_global_action' which represents the function 'app::some_global_function'.
(continues on next page)```
HPX Documentation, 1.4.1

Example:

Note The macro `HPX_PLAIN_ACTION_ID` has to be used at global namespace even if the wrapped function is located in some other namespace. The newly defined action type is placed into the global namespace as well.

Note Only one of the forms of this macro `HPX_PLAIN_ACTION` or `HPX_PLAIN_ACTION_ID` should be used for a particular action, never both.

file `applier_fwd.hpp`

```cpp
#include <hpx/config.hpp>
```

file `basename_registration_fwd.hpp`

```cpp
#include <hpx/config.hpp>
#include <hpx/components_fwd.hpp>
#include <hpx/lcos_fwd.hpp>
#include <hpx/runtime/components/make_client.hpp>
#include <hpx/runtime/naming/id_type.hpp>
#include <cstddef>
#include <string>
#include <utility>
#include <vector>
```

file `binpacking_distribution_policy.hpp`

```cpp
#include <hpx/config.hpp>
#include <hpx/assertion.hpp>
#include <hpx/dataflow.hpp>
#include <hpx/lcos/future.hpp>
#include <hpx/lcos/dataflow.hpp>
#include <hpx/performance_counters/performance_counter.hpp>
#include <hpx/runtime/components/client_base.hpp>
#include <hpx/runtime/launch_policy.hpp>
#include <hpx/functionall/bind_front.hpp>
#include <hpx/functional/bind_front.hpp>
#include <hpx/functional/bind_back.hpp>
#include <hpx/functional/invoke.hpp>
#include <hpx/functional/invoke_fused.hpp>
#include <hpx/traits/future_traits.hpp>
#include <hpx/traits/is_future.hpp>
#include <hpx/datastructures/traits/is_tuple_like.hpp>
#include <hpx/traits/pack_traversal_rebind_container.hpp>
#include <hpx/functional/result_of.hpp>
#include <hpx/functional/traits/is_callable.hpp>
#include <hpx/datastructures/detail/pack.hpp>
#include <hpx/datastructures/tuple.hpp>
#include <hpx/datastructures/traits/is_distribution_policy.hpp>
#include <hpx/datastructures/traits/is_tuple_like.hpp>
#include <hpx/datastructures/detail/pack.hpp>
#include <hpx/datastructures/tuple.hpp>
#include <hpx/traits/pack_traversal_rebind_container.hpp>
#include <hpx/functional/traits/is_callable.hpp>
#include <algorithm>
```

file `colocating_distribution_policy.hpp`

```cpp
#include <hpx/config.hpp>
#include <hpx/lcos/detail/async_colocated.hpp>
#include <hpx/lcos/async_continue_fwd.hpp>
#include <hpx/lcos/async_fwd.hpp>
#include <hpx/lcos/detail/async_colocated_fwd.hpp>
#include <hpx/lcos/future.hpp>
#include <hpx/runtime/actions/action_support.hpp>
#include <hpx/runtime/agas/primary_namespace.hpp>
#include <hpx/runtime/parcelset_fwd.hpp>
#include <hpx/runtime/agas/gva.hpp>
#include <hpx/runtime/components/make_client.hpp>
#include <hpx/runtime/naming/address.hpp>
#include <hpx/datastructures/tuple.hpp>
#include <cstddef>
#include <cstdint>
#include <string>
#include <util::unique_allocation_info.hpp>
#include <algorithm>
```

356 Chapter 2. What’s so special about HPX?
Define a component factory for a component type.

This macro is used create and to register a minimal component factory for a component type which allows it to be remotely created using the `hpx::new_<>` function.

This macro can be invoked with one, two or three arguments

**Parameters**

- **type**: The `type` parameter is a (fully decorated) type of the component type for which a factory should be defined.
- **name**: The `name` parameter specifies the name to use to register the factory. This should uniquely (system-wide) identify the component type. The `name` parameter must conform to the C++ identifier rules (without any namespace). If this parameter is not given, the first parameter is used.
- **mode**: The `mode` parameter has to be one of the defined enumeration values of the enumeration `hpx::components::factory_state_enum`. The default for this parameter is `hpx::components::factory_enabled`.

### file `component_factory.hpp`

**HPX_REGISTER_COMPONENT** (type, name, mode)

Define a component factory for a component type.

This macro is used create and to register a minimal component factory for a component type which allows it to be remotely created using the `hpx::new_<>` function.

This macro can be invoked with one, two or three arguments

**Parameters**

- **type**: The `type` parameter is a (fully decorated) type of the component type for which a factory should be defined.
- **name**: The `name` parameter specifies the name to use to register the factory. This should uniquely (system-wide) identify the component type. The `name` parameter must conform to the C++ identifier rules (without any namespace). If this parameter is not given, the first parameter is used.
- **mode**: The `mode` parameter has to be one of the defined enumeration values of the enumeration `hpx::components::factory_state_enum`. The default for this parameter is `hpx::components::factory_enabled`.

### file `copy_component.hpp`

`#include <hpx/config.hpp>`

`#include <hpx/config/warnings_prefix.hpp>`

`#include <hpx/config/warnings_suffix.hpp>`

`#include <hpx/runtime/agas/server/primary_namespace.hpp>`

`#include <hpx/runtime/actions/component_action.hpp>`

`#include <hpx/runtime/components/server/fixed_component_base.hpp>`

`#include <hpx/runtime/applier/applier.hpp>`

`#include <hpx/runtime/applier_fwd.hpp>`

`#include <hpx/runtime/components/component_type.hpp>`

`#include <hpx/runtime/naming/name.hpp>`

`#include <hpx/runtime/parcelset/parcel.hpp>`

`#include <hpx/runtime/applier/bind_naming_wrappers.hpp>`

`#include <hpx/runtime/components/server/create_component fwd.hpp>`

`#include <hpx/runtime/components_fwd.hpp>`

`#include <hpx/runtime_fwd.hpp>`

`#include <hpx/traits/is_component.hpp>`

`#include <hpx/functional/unique_function.hpp>`

`#include <sstream>`

`#include <type_traits>`

`#include <hpx/runtime/naming/id_type.hpp>`

`#include <hpx/traits/action_message_handler.hpp>`

`#include <hpx/traits/action_serialization_filter.hpp>`

`#include <hpx/allocator_support/internal_allocator.hpp>`

`#include <atomic>`

`#include <algorithm>`
<hpx/lcos/future.hpp>#include <hpx/runtime/actions/plain_action.hpp>#include <hpx/assertion.hpp>#include <hpx/preprocessor/cat.hpp>#include <hpx/preprocessor/expand.hpp>#include <hpx/preprocessor/strip_parens.hpp>#include <hpx/runtime/actions/basic_action.hpp>#include <hpx/runtime/naming/address.hpp>#include <hpx/preprocessor/nargs.hpp>#include <hpx/preprocessor/strip_parens.hpp>#include <hpx/runtime/actions/manage_object_action.hpp>#include <hpx/runtime/actions/action_support.hpp>#include <hpx/serialization/array.hpp>#include <hpx/serialization/base_object.hpp>#include <hpx/serialization/serialize_buffer.hpp>#include <hpx/static_reinit/reinitializable_static.hpp>#include <cstdlib>#include <sstream>#include <stdexcept>#include <string>#include <utility>#include <hpx/config/warnings_prefix.hpp>#include <hpx/config/warnings_suffix.hpp>#include <hpx/runtime/components/server/copy_component.hpp>#include <hpx/runtime/actions/plain_action.hpp>#include <hpx/runtime/components/stubs/runtime_support.hpp>#include <hpx/async.hpp>#include <hpx/errors.hpp>#include <hpx/lcos/detail/async_colocated_fwd.hpp>#include <hpx/runtime/actions/component_action.hpp>#include <hpx/runtime/components/server/create_component.hpp>#include <hpx/runtime/components/server/create_component_fwd.hpp>#include <hpx/runtime/components/server/component_heap.hpp>#include <hpx/datastructures/detail/pack.hpp>#include <hpx/traits/component_type_database.hpp>#include <hpx/traits/action_does_termination_detection.hpp>#include <hpx/traits/is_component.hpp>#include <atomic>#include <condition_variable>#include <list>#include <mutex>#include <set>#include <thread>#include <hpx/lcos/dataflow.hpp>#include <hpx/lcos/future.hpp>#include <hpx/lcos/packaged_action.hpp>#include <hpx/runtime/actions/action_support.hpp>#include <hpx/runtime/applier/apply.hpp>#include <hpx/runtime/components/stubs/stub_base.hpp>#include <hpx/runtime/find_here.hpp>#include <hpx/runtime/launch_policy.hpp>#include <hpx/runtime/naming/id_type.hpp>#include <hpx/runtime/naming/name.hpp>#include <hpx/traits/extract_action.hpp>#include <hpx/traits/is_distribution_policy.hpp>#include <hpx/traits/is_component.hpp>#include <atomic>#include <condition_variable>#include <list>#include <mutex>#include <set>#include <thread>#include <hpx/lcos/dataflow.hpp>#include <hpx/lcos/future.hpp>#include <hpx/lcos/packaged_action.hpp>#include <hpx/runtime/actions/action_support.hpp>#include <hpx/runtime/launch_policy.hpp>#include <hpx/runtime/naming/id_type.hpp>#include <hpx/runtime/naming/name.hpp>#include <hpx/traits/extract_action.hpp>
<hpx/traits/is_component.hpp>
#include <cstdint>
#include <memory>
#include <utility>
#include <hpx/runtime/components/target_distribution_policy.hpp>
#include <hpx/lcos/dataflow.hpp>
#include <hpx/lcos/packaged_action.hpp>
#include <hpx/runtime/actions/action_support.hpp>
#include <hpx/runtime/applier/detail/apply_implementations_fwd.hpp>
#include <hpx/runtime/components/stubs/stub_base.hpp>
#include <hpx/runtime/find_here.hpp>
#include <hpx/runtime/launch_policy.hpp>
#include <hpx/runtime/naming/id_type.hpp>
#include <hpx/serialization/serialization_fwd.hpp>
#include <hpx/traits/extract_action.hpp>
#include <hpx/traits/is_distribution_policy.hpp>
#include <hpx/traits/promise_local_result.hpp>
#include <algorithm>
#include <cstdint>
#include <memory>
#include <type_traits>
#include <vector>

#include <hpx/config.hpp>
#include <hpx/lcos/future.hpp>
#include <hpx/runtime/components/client_base.hpp>
#include <hpx/runtime/components/default_distribution_policy.hpp>
#include <hpx/lcos/dataflow.hpp>
#include <hpx/lcos/packaged_action.hpp>
#include <hpx/runtime/actions/action_support.hpp>
#include <hpx/runtime/applier/apply.hpp>
#include <hpx/runtime/components/stubs/stub_base.hpp>
#include <hpx/runtime/find_here.hpp>
#include <hpx/runtime/launch_policy.hpp>
#include <hpx/runtime/naming/id_type.hpp>
#include <hpx/runtime/naming/name.hpp>
#include <hpx/serialization/serialization_fwd.hpp>
#include <hpx/traits/extract_action.hpp>
#include <hpx/traits/is_distribution_policy.hpp>
#include <hpx/traits/promise_local_result.hpp>
#include <algorithm>
#include <cstdint>
#include <hpx/runtime/components/server/create_component.hpp>
#include <hpx/runtime/launch_policy.hpp>
#include <hpx/traits/is_client.hpp>
#include <hpx/traits/is_component.hpp>
#include <hpx/traits/lazy_enable_if.hpp>

file new.hpp
#include <hpx/config.hpp>
#include <hpx/lcos/future.hpp>
#include <hpx/runtime/components/client_base.hpp>
#include <hpx/runtime/components/default_distribution_policy.hpp>
#include <hpx/lcos/dataflow.hpp>
#include <hpx/lcos/packaged_action.hpp>
#include <hpx/runtime/actions/action_support.hpp>
#include <hpx/runtime/applier/apply.hpp>
#include <hpx/runtime/components/stubs/stub_base.hpp>
#include <hpx/runtime/find_here.hpp>
#include <hpx/runtime/launch_policy.hpp>
#include <hpx/runtime/naming/id_type.hpp>
#include <hpx/serialization/serialization_fwd.hpp>
#include <hpx/traits/extract_action.hpp>
#include <hpx/traits/is_distribution_policy.hpp>
#include <hpx/traits/promise_local_result.hpp>
#include <algorithm>
#include <cstdint>
#include <vector>

file find_here.hpp
#include <hpx/config.hpp>
#include <hpx/lcos/future.hpp>
#include <hpx/runtime/components/client_base.hpp>
#include <hpx/runtime/components/default_distribution_policy.hpp>
#include <hpx/lcos/dataflow.hpp>
#include <hpx/lcos/packaged_action.hpp>
#include <hpx/runtime/actions/action_support.hpp>
#include <hpx/runtime/applier/apply.hpp>
#include <hpx/runtime/components/stubs/stub_base.hpp>
#include <hpx/runtime/find_here.hpp>
#include <hpx/runtime/launch_policy.hpp>
#include <hpx/runtime/naming/id_type.hpp>
#include <hpx/serialization/serialization_fwd.hpp>
#include <hpx/naming/name.hpp>
#include <hpx/traits/is_distribution_policy.hpp>
#include <hpx/traits/promise_local_result.hpp>
#include <algorithm>
#include <cstdint>
#include <vector>

file find_localities.hpp
#include <hpx/config.hpp>
#include <hpx/lcos/future.hpp>
#include <hpx/runtime/components/component_type.hpp>
#include <hpx/lcos/packaged_action.hpp>
#include <hpx/runtime/actions/action_support.hpp>
#include <hpx/runtime/launch_policy.hpp>
#include <hpx/runtime/naming/id_type.hpp>
#include <vector>

file get_colocation_id.hpp
#include <hpx/errors.hpp>
#include <hpx/lcos_fwd.hpp>
#include <hpx/runtime/launch_policy.hpp>
#include <hpx/runtime/naming/id_type.hpp>
#include <vector>

file get_locality_id.hpp
#include <hpx/errors.hpp>
#include <hpx/lcos_fwd.hpp>
#include <hpx/runtime/launch_policy.hpp>
#include <hpx/runtime/naming/id_type.hpp>
#include <string>

file get_num_localities.hpp
#include <hpx/config.hpp>
#include <hpx/lcos_fwd.hpp>
#include <hpx/runtime/launch_policy.hpp>
#include <hpx/runtime/components/component_type.hpp>
#include <cstdint>

file get_os_thread_count.hpp
#include <hpx/runtime/threads/thread_data_fwd.hpp>
#include <cstdint>

file get_ptr.hpp
#include <hpx/config.hpp>
#include <hpx/assertion.hpp>
#include <hpx/runtime/agas/gva.hpp>
#include <hpx/runtime/components/component_type.hpp>
#include <hpx/runtime/naming/name.hpp>
#include <hpx/errors.hpp>
#include <boost/io/ios_state.hpp>
#include <cstdint>
#include <hpx/runtime/launch_policy.hpp>
#include <hpx/runtime/naming/address.hpp>

2.9. API reference 359
<file>get_thread_name.hpp</file>
#include <hpx/config.hpp>#include <string>

<file>get_worker_thread_num.hpp</file>
#include <hpx/config.hpp>#include <hpx/traits/component_type_is_compatible.hpp>
#include <hpx/errors.hpp>#include <cstdint>

<file>launch_policy.hpp</file>
#include <hpx/config.hpp>#include <hpx/traits/component_pin_support.hpp>
#include <hpx/traits/component_type_is_compatible.hpp>
#include <hpx/functional/bind_back.hpp>
#include <memory>

<file>unmanaged.hpp</file>
#include <hpx/runtime/naming/name.hpp>

<file>report_error.hpp</file>
#include <hpx/config.hpp>#include <cstdint>#include <exception>

<file>runtime_mode.hpp</file>
#include <hpx/config.hpp>#include <string>

<file>setParcelWriteHandler.hpp</file>
#include <hpx/config.hpp>

<file>shutdown_function.hpp</file>
#include <hpx/config.hpp>#include <hpx/functional/unix_socket.h.hpp>
#include <hpx/functional/unique_function.hpp>
#include <hpx/serialization/serialization_fwd.hpp>
#include <type_traits>
#include <utility>

<file>startup_function.hpp</file>
#include <hpx/config.hpp>#include <hpx/functional/unix_socket.h.hpp>
#include <hpx/config.hpp>

<file>scheduler_mode.hpp</file>
#include <cstdint>

<file>thread_data fwd.hpp</file>
#include <hpx/config.hpp>#include <hpx/coroutines/coroutine_fwd.hpp>
#include <hpx/coroutines/thread_enums.hpp>
#include <hpx/coroutines/thread_id_type.hpp>
#include <hpx/serialization/serialization_fwd.hpp>
#include <type_traits>
#include <util_fwd.hpp>
#include <cstdint>
#include <memory>
#include <utility>

<file>thread_helpers.hpp</file>
#include <hpx/config.hpp>#include <hpx/basic_execution/register_locks.hpp>
#include <hpx/coroutines/thread_enums.hpp>
#include <hpx/coroutines/thread_id_type.hpp>
#include <hpx/functional/unix_socket.h.hpp>
#include <hpx/serialization/serialization_fwd.hpp>
#include <type_traits>
#include <util_fwd.hpp>
#include <cstdint>
#include <memory>
#include <utility>

<file>thread_pool_base.hpp</file>
#include <hpx/config.hpp>
#include <hpx/concurrency/barrier.hpp>
#include <hpx/runtime/thread_pool_helpers.hpp>
#include <hpx/runtime/threads/detail/network_background_callback.hpp>
#include <hpx/topology/cpu_mask.hpp>
#include <hpx/topology/topology.hpp>
#include <hpx/timing/steady_clock.hpp>
#include <hpx/state.hpp>
#include <hpx/util_fwd.hpp>
#include <thread>
#include <cstdint>
#include <memory>
#include <util_fwd.hpp>
#include <exception>
#include <functional>

Chapter 2. What’s so special about HPX?
2.9.2 Modules reference

datastructures

```cpp
#include <compatibility/hpx/util/any.hpp>
#include <compatibility/hpx/util/optional.hpp>
#include <compatibility/hpx/util/tuple.hpp>
#include <compatibility/hpx/traits/supports_streaming_with_any.hpp>
#include <compatibility/hpx/traits/is_tuple_like.hpp>
#include <hpx/datastructures/any.hpp>
```

template <>
template<>

class basic_any<void, void, void, std::true_type>

Public Functions

```cpp
basic_any()  

basic_any(basic_any const &x)  

basic_any(basic_any &&x)  

template <typename T, typename Enable = typename std::enable_if<!std::is_same<basic_any, typename std::decay<T>::type>::value>::type>

basic_any(T &&x, typename std::enable_if<std::is_copy_constructible<typename std::decay<T>::type>::value>::type * = nullptr)  

basic_any()  

basic_any &operator=(basic_any const &x)  

basic_any &operator=(basic_any &&rhs)  

template <typename T, typename Enable = typename std::enable_if<!std::is_same<basic_any, typename std::decay<T>::type>::value>::type>

basic_any &operator=(T &&rhs)  

basic_any &swap(basic_any &x)  

std::type_info const &type() const  

template <typename T>
T const &cast() const  

bool has_value() const  

void reset()  

bool equal_to(basic_any const &rhs) const
```
Private Functions

basic_any &assign (basic_any const &x)

Private Members

detail::any::fxn_ptr_table<void, void, void, std::true_type> *table
void *object

Private Static Functions

template<typename T, typename... Ts>
static void new_object (void *object, std::true_type, Ts&&... ts)

template<typename T, typename... Ts>
static void new_object (void *object, std::false_type, Ts&&... ts)

template<typename Char>

class basic_any<void, void, Char, std::true_type>

Public Functions

basic_any ()

basic_any (basic_any const &x)

basic_any (basic_any &&x)

template<typename T, typename Enable = typename std::enable_if<!std::is_same<basic_any, typename std::decay<T>::type>::value>::type>
basic_any (T &&x, typename std::enable_if < std::is_copy_constructible<typename std::decay<T>::type>::value >::type * = nullptr)

~basic_any ()

basic_any &operator= (basic_any const &x)

basic_any &operator= (basic_any &&rhs)

template<typename T, typename Enable = typename std::enable_if< !std::is_same<basic_any, typename std::decay<T>::type>::value && std::is_copy_constructible<typename std::decay<T>::type>::value>::type>
basic_any &operator= (T &&rhs)

basic_any &swap (basic_any &x)

std::type_info const &type () const

template<typename T>
T const &cast () const

bool has_value () const

void reset ()

bool equal_to (basic_any const &rhs) const

Private Functions

basic_any &assign (basic_any const &x)
Private Members

detail::any::fxn_ptr_table<void, void, Char, std::true_type> *table
void *object

Private Static Functions

template <typename T, typename... Ts>
static void new_object (void *object, std::true_type, Ts&&... ts)
template <typename T, typename... Ts>
static void new_object (void *object, std::false_type, Ts&&... ts)

template <>
template<>
class basic_any<void, void, void, std::false_type>

Public Functions

basic_any ()
basic_any (basic_any &&x)

//template <typename T, typename Enable = typename std::enable_if<!std::is_same<basic_any, typename std::decay<T>::type>::value>::type>
basic_any (T &&x, typename std::enable_if<std::is_move_constructible<typename std::decay<T>::type>::value>::type * = nullptr)

//template <typename T, typename Enable = typename std::enable_if< !std::is_same<basic_any, typename std::decay<T>::type>::value && std::is_move_constructible< typename std::decay<T>::type>::value>::type>
basic_any (basic_any const &&x)
basic_any &operator= (basic_any const &x)

~basic_any ()
basic_any &operator= (basic_any &&rhs)

//template <typename T, typename Enable = typename std::enable_if< !std::is_same<basic_any, typename std::decay<T>::type>::value>::type>
basic_any &operator= (T &rhs)
basic_any &swap (basic_any &x)

std::type_info const &type () const

template <typename T>
T const &cast () const

bool has_value () const

void reset ()

bool equal_to (basic_any const &rhs) const

Private Members

detail::any::fxn_ptr_table<void, void, void, std::false_type> *table
void *object
Private Static Functions

```cpp
template <typename T, typename... Ts>
static void new_object (void *&object, std::true_type, Ts&&... ts)
```

```cpp
template <typename T, typename... Ts>
static void new_object (void *&object, std::false_type, Ts&&... ts)
```

template <typename Char>
template<>
class basic_any<void, void, Char, std::false_type>

Public Functions

```cpp
basic_any ()
```

```cpp
basic_any (basic_any &&x)
```

```cpp
basic_any (basic_any const &x)
```

```cpp
basic_any &operator= (basic_any const &x)
```

```cpp
basic_any &operator= (basic_any &&rhs)
```

```cpp
template <typename T, typename> Enable = typename std::enable_if<!std::is_same<basic_any, typename std::decay<T>::type>::value>::type *
```

```cpp
basic_any (basic_any const &x)
```

```cpp
std::type_info const &type () const
```

```cpp
template <typename T>
T const &cast () const
```

```cpp
bool has_value () const
```

```cpp
void reset ()
```

```cpp
bool equal_to (basic_any const &rhs) const
```

Private Members

detail::any::fxn_ptr_table<void, void, Char, std::false_type> *table

void *object

Private Static Functions

```cpp
template <typename T, typename... Ts>
static void new_object (void *&object, std::true_type, Ts&&... ts)
```

```cpp
template <typename T, typename... Ts>
static void new_object (void *&object, std::false_type, Ts&&... ts)
```
namespace hpx

namespace util

Typedefs

using hpx::util::any_nonser = typedef basic_any<void, void, void, std::true_type>
using hpx::util::streamable_any_nonser = typedef basic_any<void, void, char, std::true_type>
using hpx::util::streamable_wany_nonser = typedef basic_any<void, void, wchar_t, std::true_type>
using hpx::util::unique_any_nonser = typedef basic_any<void, void, void, std::false_type>
using hpx::util::streamable_unique_any_nonser = typedef basic_any<void, void, char, std::false_type>
using hpx::util::streamable_unique_wany_nonser = typedef basic_any<void, void, wchar_t, std::false_type>

Functions

template <typename IArch, typename OArch, typename Char, typename Copyable, typename Enable = typename std::enable_if<!std::is_void<Char>::value>::type>
std::basic_istream<Char> &
operator>>(std::basic_istream<Char> &i, basic_any<IArch, OArch, Char, Copyable> &obj)

template <typename IArch, typename OArch, typename Char, typename Copyable, typename Enable = typename std::enable_if<!std::is_void<Char>::value>::type>
std::basic_ostream<Char> &
operator<<(std::basic_ostream<Char> &o, basic_any<IArch, OArch, Char, Copyable> const &obj)

void swap(basic_any<IArch, OArch, Char, Copyable> &lhs, basic_any<IArch, OArch, Char, Copyable> &rhs)

template <typename T, typename IArch, typename OArch, typename Char, typename Copyable>
T *any_cast(basic_any<IArch, OArch, Char, Copyable> *operand)

template <typename T, typename IArch, typename OArch, typename Char, typename Copyable>
T const *any_cast(basic_any<IArch, OArch, Char, Copyable> const *operand)

template <typename T, typename IArch, typename OArch, typename Char, typename Copyable>
T any_cast(basic_any<IArch, OArch, Char, Copyable> &operand)

template <typename T, typename IArch, typename OArch, typename Char, typename Copyable>
T const &any_cast(basic_any<IArch, OArch, Char, Copyable> const &operand)

template <typename T>
basic_any<void, void, void, std::true_type> make_any_nonser(T &&t)

template <typename T, typename Char>
basic_any<void, void, Char, std::true_type> make_streamable_any_nonser(T &&t)

template <typename T>
basic_any<void, void, void, std::false_type> make_unique_any_nonser(T &&t)

template <typename T, typename Char>
basic_any<void, void, Char, std::false_type> make_streamable_unique_any_nonser(T &&t)

struct bad_any_cast : public bad_cast
Public Functions

`bad_any_cast (std::type_info const &src, std::type_info const &dest)`

`const char *what () const`

Public Members

`const char *from`

`const char *to`

```cpp
template <typename Char>
class basic_any<void, void, Char, std::false_type>
```

Public Functions

```cpp
basic_any ()
```

```cpp
basic_any (basic_any &&x)
```

```cpp
template <typename T, typename Enable = typename std::enable_if<!std::is_same<basic_any, typename std::decay<T>::type>::value>::type>
basic_any (T &&x, typename std::enable_if<std::is_move_constructible<typename std::decay<T>::type>::value>::type * = nullptr)
```

```cpp
basic_any (basic_any const &x)
```

```cpp
basic_any &operator= (basic_any const &x)
```

```cpp
~basic_any ()
```

```cpp
basic_any &operator= (basic_any &&rhs)
```

```cpp
template <typename T, typename Enable = typename std::enable_if<!std::is_same<basic_any, typename std::decay<T>::type>::value && std::is_move_constructible<typename std::decay<T>::type>::value>::type>
basic_any &operator= (T &&rhs)
```

```cpp
basic_any &swap (basic_any &x)
```

```cpp
std::type_info const &type () const
```

```cpp
template <typename T>
T const &cast () const
```

```cpp
bool has_value () const
```

```cpp
void reset ()
```

```cpp
bool equal_to (basic_any const &rhs) const
```

Private Members

detail::any::fxn_ptr_table<void, void, Char, std::false_type> *table

```cpp
void *object
```
Private Static Functions

\texttt{template <typename T, typename... Ts> static void new\_object (void \&* object, std::true\_type, Ts&&... ts)}
\texttt{template <typename T, typename... Ts> static void new\_object (void \&* object, std::false\_type, Ts&&... ts)}

\texttt{template <typename Char> template<>
class basic\_any< void, void, Char, std::true\_type>}

Public Functions

\texttt{basic\_any ()}
\texttt{basic\_any (basic\_any const \&x)}
\texttt{basic\_any (basic\_any &&x)}
\texttt{template <typename T, typename Enable = typename std::enable\_if< !std::is\_same<basic\_any, typename std::decay<T>::type>::value>::type>
basic\_any (T &&x, typename std::enable\_if< std::is\_copy\_constructible<typename std::decay<T>::type>::value>::type \*= nullptr)}
\texttt{~basic\_any ()}
\texttt{basic\_any \&operator= (basic\_any const \&x)}
\texttt{basic\_any \&operator= (basic\_any &&rhs)}
\texttt{template <typename T, typename Enable = typename std::enable\_if< !std::is\_same<basic\_any, typename std::decay<T>::type>::value>::type>
basic\_any \&operator= (T &&rhs)}
\texttt{basic\_any \&swap (basic\_any \&x)}
\texttt{std::type\_info const \&type () const}
\texttt{template <typename T> T const \&cast () const}
\texttt{bool has\_value () const}
\texttt{void reset ()}
\texttt{bool equal\_to (basic\_any const \&rhs) const}

Private Functions

\texttt{basic\_any \&assign (basic\_any const \&x)}

Private Members

detail::any::fxn\_ptr\_table<void, void, Char, std::true\_type> \*table
\texttt{void \*object}
Private Static Functions

template <typename T, typename... Ts>
static void new_object (void * & object, std::true_type, Ts &... ts)

template <typename T, typename... Ts>
static void new_object (void * & object, std::false_type, Ts &... ts)

template <>
template>
class basic_any< void, void, void, std::false_type >

Public Functions

basic_any()

basic_any (basic_any & & x)

template <typename T, typename > Enable = typename std::enable_if<!std::is_same<basic_any, typename std::decay<T>::type>::value>::type
basic_any (T & & x, typename std::enable_if<!std::is_move_constructible<typename std::decay<T>::type>::value>::type * = nullptr)

basic_any (basic_any const & x)

basic_any & operator = (basic_any const & x)

~basic_any()

basic_any & operator = (basic_any & & rhs)

namespace std
{ type_info const & type () const

template <typename T>
T const & cast () const

bool has_value () const

void reset ()

bool equal_to (basic_any const & rhs) const

Private Members

detail::any::fxn_ptr_table<void, void, void, std::false_type> * table

void * object

Private Static Functions

template <typename T, typename... Ts>
static void new_object (void * & object, std::true_type, Ts & ... ts)

template <typename T, typename... Ts>
static void new_object (void * & object, std::false_type, Ts & ... ts)
template <>
template<>
class basic_any<void, void, void, std::true_type>

Public Functions

basic_any()

basic_any(basic_any const &x)

basic_any(basic_any &&x)

basic_any(T &&x, typename std::enable_if<!std::is_same<basic_any, typename std::decay<T>::type>::value>::type * = nullptr)

~basic_any()

basic_any &operator=(basic_any const &x)

basic_any &operator=(basic_any &&rhs)

basic_any &swap(basic_any &x)

std::type_info const &type() const

template <typename T>
T const &cast() const

bool has_value() const

void reset()

bool equal_to(basic_any const &rhs) const

Private Functions

basic_any &assign(basic_any const &x)

Private Members

detail::any::fxn_ptr_table<void, void, void, std::true_type> *table

void *object

Private Static Functions

template <typename T, typename... Ts>
static void new_object(void *object, std::true_type, Ts&&... ts)

template <typename T, typename... Ts>
static void new_object(void *object, std::false_type, Ts&&... ts)
#include <hpx/datastructures/optional.hpp>

```cpp
template <typename T>
template<>
struct hash<hpx::util::optional<T>>
```

## Public Functions

```cpp
std::size_t operator()(hpx::util::optional<T> const &arg) const
```

```cpp
namespace hpx
```

```cpp
namespace util
```

## Functions

```cpp
template <typename T>
bool operator==(optional<T> const &lhs, optional<T> const &rhs)
```

```cpp
template <typename T>
bool operator!=(optional<T> const &lhs, optional<T> const &rhs)
```

```cpp
template <typename T>
bool operator<(optional<T> const &lhs, optional<T> const &rhs)
```

```cpp
template <typename T>
bool operator<=(optional<T> const &lhs, optional<T> const &rhs)
```

```cpp
template <typename T>
bool operator>(optional<T> const &lhs, optional<T> const &rhs)
```

```cpp
template <typename T>
bool operator>=(optional<T> const &lhs, optional<T> const &rhs)
```

```cpp
template <typename T>
bool operator==(optional<T> const &opt, nullopt_t)
```

```cpp
template <typename T>
bool operator==(nullopt_t, optional<T> const &opt)
```

```cpp
template <typename T>
bool operator!=(optional<T> const &opt, nullopt_t)
```

```cpp
template <typename T>
bool operator!=(nullopt_t, optional<T> const &opt)
```

```cpp
template <typename T>
bool operator<(optional<T> const &opt, nullopt_t)
```

```cpp
template <typename T>
bool operator<(nullopt_t, optional<T> const &opt)
```

```cpp
template <typename T>
bool operator>=(optional<T> const &opt, nullopt_t)
```

```cpp
template <typename T>
bool operator>=(nullopt_t, optional<T> const &opt)
```

```cpp
template <typename T>
bool operator>(optional<T> const &opt, nullopt_t)
```

```cpp
template <typename T>
bool operator>(nullopt_t, optional<T> const &opt)
```
template<typename T>
bool operator>(nullopt_t, optional<T> const &opt)

template<typename T>
bool operator<=(optional<T> const &opt, nullopt_t)

template<typename T>
bool operator<=(nullopt_t, optional<T> const &opt)

template<typename T>
bool operator==(optional<T> const &opt, T const &value)

template<typename T>
bool operator==(T const &value, optional<T> const &opt)

template<typename T>
bool operator<(optional<T> const &opt, T const &value)

template<typename T>
bool operator<(T const &value, optional<T> const &opt)

template<typename T>
bool operator>=(optional<T> const &opt, T const &value)

template<typename T>
bool operator>=(T const &value, optional<T> const &opt)

template<typename T>
bool operator>(optional<T> const &opt, T const &value)

template<typename T>
bool operator>(T const &value, optional<T> const &opt)

template<typename T>
bool operator>=(optional<T> const &opt, T const &value)

template<typename T>
bool operator>=(T const &value, optional<T> const &opt)

template<typename T>
void swap(optional<T> &x, optional<T> &y)

template<typename T>
optional<typename std::decay<T>::type> make_optional(T &&v)

template<typename T, typename... Ts>
optional<T> make_optional(Ts&&... ts)

template<typename T, typename... Ts>
optional<T, typename... Ts> make_optional(std::initializer_list<Ts> il, Ts&&... ts)

Variables

nullopt_t nullopt = {nullopt_t::init()}

class bad_optional_access : public logic_error
Public Functions

`bad_optional_access (std::string const &what_arg)`

`bad_optional_access (char const *what_arg)`

`struct nullopt_t`

Public Functions

`nullopt_t (nullopt_t::init)`

`template <typename T>
class optional`

Public Types

`template<>
using value_type = T`

Public Functions

`optional ()`

`optional (nullopt_t)`

`optional (optional const &other)`

`optional (optional &&other)`

`optional (T const &val)`

`optional (T &&val)`

`template <typename... Ts>
optional (in_place_t, Ts&&... ts)`

`template <typename U, typename... Ts>
optional (in_place_t, std::initializer_list<U> il, Ts&&... ts)`

`optional ()`

`optional &operator=(optional const &other)`

`optional &operator=(optional &&other)`

`optional &operator=(T const &other)`

`optional &operator=(T &&other)`

`optional &operator=(nullopt_t)`

`T const *operator-> () const`

`T *operator-> ()`

`T const &operator*() const`

`T &operator*() const`
operator bool() const
bool has_value() const
T &value()
T const &value() const

template <typename U>
HPX_CXX14_CONSTEXPR T hpx::util::optional::value_or(U && value) const

template <typename... Ts>
void emplace(Ts&&... ts)

void swap(optional & other)
void reset()

Private Members

std::aligned_storage<sizeof(T), alignof(T)>::type storage_
bool empty_

namespace _optional_swap

Functions

template <typename T>
void check_swap()

namespace std


template <typename T>
template<>
struct hash<hpx::util::optional<T>>

Public Functions

std::size_t operator() (hpx::util::optional<T> const & arg) const

#include <hpx/datastructures/tuple.hpp>

template <typename T0, typename T1>
template<>
struct tuple_element<0, std::pair<T0, T1> >

Public Types

template<>
using type = T0
Public Static Functions

static HPX_HOST_DEVICE type& hpx::util::tuple_element::get(std::pair< T0, T1 > & tuple)
static HPX_HOSTDEVICE type const& hpx::util::tuple_element::get(std::pair< T0, T1 > const & tuple)

template <typename T0, typename T1>
template<>
struct tuple_element<1, std::pair<T0, T1>>

Public Types

template<>
using type = T1

Public Static Functions

static HPX_HOST_DEVICE type& hpx::util::tuple_element::get(std::pair< T0, T1 > & tuple)
static HPX_HOSTDEVICE type const& hpx::util::tuple_element::get(std::pair< T0, T1 > const & tuple)

template <std::size_t I, typename Type, std::size_t Size>
template<>
struct tuple_element<I, boost::array<Type, Size>>

Public Types

template<>
using type = Type

Public Static Functions

static HPX_HOST_DEVICE type& hpx::util::tuple_element::get(boost::array< Type, Size > & tuple)
static HPX_HOSTDEVICE type const& hpx::util::tuple_element::get(boost::array< Type, Size > const & tuple)

template <std::size_t I, typename Type, std::size_t Size>
template<>
struct tuple_element<I, std::array<Type, Size>>

Public Types

template<>
using type = Type

Public Static Functions

static HPX_HOST_DEVICE type& hpx::util::tuple_element::get(std::array< Type, Size > & tuple)
static HPX_HOSTDEVICE type const& hpx::util::tuple_element::get(std::array< Type, Size > const & tuple)

namespace hpx

namespace util
Functions

```cpp
template <typename... Ts>
HPX_HOST_DEVICE tuple<typename decay_unwrap<Ts>::type...> hpx::util::make_tuple(Ts &&... vs)

template <typename... Ts>
HPX_HOST_DEVICE tuple<Ts&&...> hpx::util::forward_as_tuple(Ts &&... vs)

template <typename... Ts>
HPX_HOST_DEVICE tuple<Ts&...> hpx::util::tie(Ts &... vs)

template <typename... Tuples>
HPX_HOST_DEVICE auto hpx::util::tuple_cat(Tuples &&... tuples)
```

```cpp
template <typename... Ts, typename... Us>
HPX_HOST_DEVICE std::enable_if<sizeof...(Ts) == sizeof...(Us), bool>::type hpx::util::operator==(tuple < Ts... > const & t, tuple < Us... > const & u)

template <typename... Ts, typename... Us>
HPX_HOST_DEVICE std::enable_if<sizeof...(Ts) == sizeof...(Us), bool>::type hpx::util::operator!=(tuple < Ts... > const & t, tuple < Us... > const & u)

template <typename... Ts, typename... Us>
HPX_HOST_DEVICE std::enable_if<sizeof...(Ts) == sizeof...(Us), bool>::type hpx::util::operator<(tuple < Ts... > const & t, tuple < Us... > const & u)

template <typename... Ts, typename... Us>
HPX_HOST_DEVICE std::enable_if<sizeof...(Ts) == sizeof...(Us), bool>::type hpx::util::operator>(tuple < Ts... > const & t, tuple < Us... > const & u)

template <typename... Ts, typename... Us>
HPX_HOST_DEVICE std::enable_if<sizeof...(Ts) == sizeof...(Us), bool>::type hpx::util::operator<=(tuple < Ts... > const & t, tuple < Us... > const & u)

template <typename... Ts, typename... Us>
HPX_HOST_DEVICE std::enable_if<sizeof...(Ts) == sizeof...(Us), bool>::type hpx::util::operator>=(tuple < Ts... > const & t, tuple < Us... > const & u)

template <typename... Ts>
HPX_HOST_DEVICE void hpx::util::swap(tuple < Ts... > & x, tuple < Ts... > & y)
```

Variables

```cpp
detail::ignore_type const ignore = {}
```

```cpp
template <typename... Ts>

class tuple
```

Public Functions

```cpp
template <typename Dependent = void, typename Enable = typename std::enable_if<hpx::util::detail::all_of<std::is_constructible<Ts>...>::value, Dependent>::type>
HPX_HOST_DEVICE tuple ()
```

```cpp
HPX_HOST_DEVICE tuple (Ts const&... vs)
```

```cpp
template <typename U, typename... Us, typename Enable = typename std::enable_if<
detail::pack<U, Us...>::size == detail::pack<Ts...>::size && std::conditional<
detail::pack<Us...>::size == 0, typename std::enable_if<
!std::is_same<
tuple, typename std::decay<U>::type>::value && 
!detail::are_tuples_compatible_not_same<
tuple, U&&>::value, detail::are_tuples_compatible<
tuple, tuple<U&&>>::type>::value>::type>::value::type>
HPX_HOST_DEVICE tuple (U &&v, Us&&... vs)
```

```
tuple (tuple const&)
tuple (tuple&&)
```

```cpp
template <typename UTuple, typename Enable = typename std::enable_if<
detail::are_tuples_compatible_not_same<
tuple, UTuple&&>::value>::type>
HPX_HOST_DEVICE tuple (UTuple &&other)
```

```cpp
HPX_HOST_DEVICE tuple& hpx::util::tuple::operator=(tuple const & other)
HPX_HOST_DEVICE tuple& hpx::util::tuple::operator=(tuple && other)
```
template <typename UTuple>
HPX_HOST_DEVICE std::enable_if< tuple_size<typename std::decay<UTuple>::type>::value == detail::pack<Ts...>::size, tuple&>::type hpx::util::tuple::operator=(UTuple && other)

Public Members

detail::tuple_impl< typename detail::make_index_pack<sizeof...(Ts)>::type, Ts... > _impl

template <>
template<>
class tuple<>

Public Functions

HPX_HOST_DEVICE tuple ()
HPX_HOST_DEVICE tuple (tuple const &)
HPX_HOST_DEVICE tuple (tuple &&)
HPX_HOST_DEVICE tuple& hpx::util::tuple::operator=(tuple const &)
HPX_HOST_DEVICE tuple& hpx::util::tuple::operator=(tuple &&)
HPX_HOST_DEVICE void hpx::util::tuple::swap(tuple &)

template <typename T0, typename T1>
template<>
struct tuple_element<0, std::pair<T0, T1>>

Public Types

template<>
using type = T0

Public Static Functions

static HPX_HOST_DEVICE type& hpx::util::tuple_element::get(std::pair< T0, T1 > &)
static HPX_HOST_DEVICE type const& hpx::util::tuple_element::get(std::pair< T0, T1 > const &)

template <typename T0, typename T1>
template<>
struct tuple_element<1, std::pair<T0, T1>>

Public Types

template<>
using type = T1

Public Static Functions

static HPX_HOST_DEVICE type& hpx::util::tuple_element::get(std::pair< T0, T1 > &)
static HPX_HOST_DEVICE type const& hpx::util::tuple_element::get(std::pair< T0, T1 > const &)

template <std::size_t I, typename Type, std::size_t Size>
template<>
struct tuple_element<I, boost::array<Type, Size>>

Public Types

template<>
using type = Type

Public Static Functions

static HPX_HOST_DEVICE type& hpx::util::tuple_element::get(boost::array<Type, Size> & tuple)
static HPX_HOST_DEVICE type const& hpx::util::tuple_element::get(boost::array<Type, Size> const & tuple)

template< std::size_t I, typename Type, std::size_t Size >

template<>
struct tuple_element<I, std::array<Type, Size>>

Public Types

template<>
using type = Type

Public Static Functions

static HPX_HOST_DEVICE type& hpx::util::tuple_element::get(std::array<Type, Size> & tuple)
static HPX_HOST_DEVICE type const& hpx::util::tuple_element::get(std::array<Type, Size> const & tuple)

template< std::size_t I, typename... Ts >

template<>
struct tuple_element<I, tuple<Ts...>>

Public Types

template<>
using type = typename detail::at_index::type

Public Static Functions

static HPX_HOST_DEVICE type& hpx::util::tuple_element::get(tuple<Ts...> & tuple)
static HPX_HOST_DEVICE type const& hpx::util::tuple_element::get(tuple<Ts...> const & tuple)

template<class T>
struct tuple_size
  Subclassed by hpx::util::tuple_size< const T >, hpx::util::tuple_size< const volatile T >,
  hpx::util::tuple_size< volatile T >

namespace adl_barrier

Functions

template <std::size_t I, typename Tuple>
HPX_HOST_DEVICE tuple_element<I, Tuple>::type & hpx::util::adl_barrier::get(Tuple & tuple)
template <std::size_t I, typename Tuple>
HPX_HOST_DEVICE tuple_element<I, Tuple>::type const & hpx::util::adl_barrier::get(Tuple const & tuple)
HPX HOST_DEVICE tuple_element< I, Tuple >::type const & hpx::util::adl_barrier::get(Tuple const & t)

template <std::size_t I, typename Tuple>
HPX_HOST_DEVICE tuple_element< I, Tuple >::type && hpx::util::adl_barrier::get(Tuple && t)

template <std::size_t I, typename Tuple>
HPX_HOST_DEVICE tuple_element< I, Tuple >::type const && hpx::util::adl_barrier::get(Tuple const && t)

#include <hpx/datastructures/traits/supports_streaming_with_any.hpp>

#include <hpx/datastructures/traits/is_tuple_like.hpp>

namespace hpx

namespace traits

    template <typename T>
    struct is_tuple_like: public hpx::traits::detail::is_tuple_like_impl<std::remove_cv<T>::type>
    // #include <is_tuple_like.hpp> Deduces to a true type if the given parameter T has a specific tuple like size.

resiliency

#include <hpx/resiliency/dataflow_replicate.hpp>

namespace hpx

namespace resiliency

Functions

    template <typename Vote, typename Pred, typename F, typename... Ts>
    hpx::future<typename hpx::util::detail::invoke_deferred_result<F, Ts...>::type>
dataflow_replicate_vote_validate(
        std::size_t n,
        Vote && vote,
        Pred && pred,
        F && f,
        Ts &&... ts)

    Launch given function f exactly n times. Run all the valid results against a user provided voting function. Return the valid output.

    Delay the invocation of f if any of the arguments to f are futures.

    template <typename Vote, typename F, typename... Ts>

2.9. API reference 379
hpx::future<typename hpx::util::detail::invoke_deferred_result<F, Ts...>::type> dataflow_replicate_vote (std::size_t n, Vote &&vote, F &&f, Ts&&... ts)

Launch given function \( f \) exactly \( n \) times. Run all the valid results against a user provided voting function. Return the valid output.

Delay the invocation of \( f \) if any of the arguments to \( f \) are futures.

template <typename Pred, typename F, typename... Ts>

hpx::future<typename hpx::util::detail::invoke_deferred_result<F, Ts...>::type> dataflow_replicate_validate (std::size_t n, Pred &&pred, F &&f, Ts&&... ts)

Launch given function \( f \) exactly \( n \) times. Verify the result of those invocations using the given predicate \( pred \). Return the first valid result.

Delay the invocation of \( f \) if any of the arguments to \( f \) are futures.

template <typename F, typename... Ts>

hpx::future<typename hpx::util::detail::invoke_deferred_result<F, Ts...>::type> dataflow_replicate (std::size_t n, F &&f, Ts&&... ts)

Launch given function \( f \) exactly \( n \) times. Return the first valid result.

Delay the invocation of \( f \) if any of the arguments to \( f \) are futures.

#include <hpx/resiliency/async_replicate.hpp>

namespace hpx

namespace resiliency

Functions

template <typename Vote, typename Pred, typename F, typename... Ts>

hpx::future<typename hpx::util::detail::invoke_deferred_result<F, Ts...>::type> async_replicate_vote_validate (std::size_t n, Vote &&vote, Pred &&pred, F &&f, Ts&&... ts)
Asynchronously launch given function \( f \) exactly \( n \) times. Verify the result of those invocations using the given predicate \( \text{pred} \). Run all the valid results against a user provided voting function. Return the valid output.

```cpp
template <typename Vote, typename F, typename... Ts>
hp::future<typename hp::util::detail::invoke_deferred_result<F, Ts...>::type> async_replicate_vote (std::size_t n, Vote &&vote, F &&f, Ts&&... ts)
```

Asynchronously launch given function \( f \) exactly \( n \) times. Verify the result of those invocations using the given predicate \( \text{pred} \). Run all the valid results against a user provided voting function. Return the valid output.

```cpp
template <typename Pred, typename F, typename... Ts>
hp::future<typename hp::util::detail::invoke_deferred_result<F, Ts...>::type> async_replicate_validate (std::size_t n, Pred &&pred, F &&f, Ts&&... ts)
```

Asynchronously launch given function \( f \) exactly \( n \) times. Verify the result of those invocations by checking for exception. Return the first valid result.

```cpp
template <typename F, typename... Ts>
hp::future<typename hp::util::detail::invoke_deferred_result<F, Ts...>::type> async_replicate (std::size_t n, F &&f, Ts&&... ts)
```

Asynchronously launch given function \( f \) exactly \( n \) times. Verify the result of those invocations by checking for exception. Return the first valid result.

```cpp
namespace functional

Functional version of \( \text{hpx::resiliency::async_replay} \).

Functional version of \( \text{hpx::resiliency::async_replicate_validate} \) and \( \text{hpx::resiliency::async_replicate} \)

struct async_replicate

Public Functions

```cpp
template <typename F, typename... Ts>
auto operator() (std::size_t n, F &&f, Ts&&... ts) const
```

```cpp
struct async_replicate_validate

Public Functions

```cpp
template <typename Pred, typename F, typename... Ts>
auto operator() (std::size_t n, Pred &&pred, F &&f, Ts&&... ts) const
```
struct async_replicate_vote

Public Functions

template <typename Vote, typename F, typename... Ts>
auto operator() (std::size_t n, Vote &&vote, F &&f, Ts&&... ts) const

struct async_replicate_vote_validate

Public Functions

template <typename Vote, typename Pred, typename F, typename... Ts>
auto operator() (std::size_t n, Vote &&vote, Pred &&pred, F &&f, Ts&&... ts) const

#include <hpx/resiliency/resiliency.hpp>
#include <hpx/resiliency/force_linking.hpp>

namespace hpx

namespace resiliency

Functions

force_linking_helper &force_linking()

struct force_linking_helper

Public Members

unsigned int (*major_version)()
unsigned int (*minor_version)()
unsigned int (*subminor_version)()
unsigned long (*full_version)()
std::string (*full_version_str)()

#include <hpx/resiliency/version.hpp>

Defines

HPX_RESILIENCY_VERSION_FULL
HPX_RESILIENCY_VERSION_MAJOR
HPX_RESILIENCY_VERSION_MINOR
HPX_RESILIENCY_VERSION_SUBMINOR
HPX_RESILIENCY_VERSION_DATE

namespace hpx

namespace resiliency
**Functions**

```cpp
unsigned int major_version()
unsigned int minor_version()
unsigned int subminor_version()
unsigned long full_version()
std::string full_version_str()
```

```cpp
#include <hpx/resiliency/config.hpp>
#include <hpx/resiliency/dataflow_replay.hpp>

namespace hpx

namespace resiliency

**Functions**

```cpp
template <typename Pred, typename F, typename... Ts>
hpx::future<typename hpx::util::detail::invoke_deferred_result<F, Ts...>::type>
dataflow_replay_validate (std::size_t n, Pred &&pred, F &&f, Ts&&... ts)
```

Asynchronously launch given function `f`. Verify the result of those invocations using the given predicate `pred`. Repeat launching on error exactly `n` times.

```cpp
template <typename F, typename... Ts>
hpx::future<typename hpx::util::detail::invoke_deferred_result<F, Ts...>::type>
dataflow_replay (std::size_t n, F &&f, Ts&&... ts)
```

Asynchronously launch given function `f`. Repeat launching on error exactly `n` times.

Delay the invocation of `f` if any of the arguments to `f` are futures.
Asynchronously launch given function \( f \). Verify the result of those invocations using the given predicate \( \text{pred} \). Repeat launching on error exactly \( n \) times (except if \text{abort_replay_exception} is thrown).

Asynchronously launch given function \( f \). Repeat launching on error exactly \( n \) times (except if \text{abort_replay_exception} is thrown).

namespace functional

Functional version of \( \text{hpx::resiliency::async_replay} \).

Functional version of \( \text{hpx::resiliency::async_replicate_validate} \) and \( \text{hpx::resiliency::async_replicate} \)

struct async_replay

Public Functions

\begin{verbatim}
template <typename F, typename... Ts>
auto operator() (std::size_t n, F &f, Ts&&... ts) const
\end{verbatim}

struct async_replay_validate

Public Functions

\begin{verbatim}
template <typename Pred, typename F, typename... Ts>
auto operator() (std::size_t n, Pred &pred, F &f, Ts&&... ts) const
\end{verbatim}

basic_execution

#include <compatibility/hpx/util/register_locks.hpp>

#include <hpx/basic_execution.hpp>

#include <hpx/basic_execution/this_thread.hpp>

namespace hpx

    namespace basic_execution
namespace this_thread

## Functions

```cpp
hpx::basic_execution::agent_ref agent ()
```

```cpp
void yield (char const *desc = "hpx::basic_execution::this_thread::yield")
```

```cpp
void yield_k (std::size_t k, char const *desc = "hpx::basic_execution::this_thread::yield_k")
```

```cpp
void suspend (char const *desc = "hpx::basic_execution::this_thread::suspend")
```

```cpp
template <typename Rep, typename Period>
void sleep_for (std::chrono::duration<Rep, Period> const &sleep_duration, char const *desc = "hpx::basic_execution::this_thread::sleep_for")
```

```cpp
template <class Clock, class Duration>
void sleep_until (std::chrono::time_point<Clock, Duration> const &sleep_time, char const *desc = "hpx::basic_execution::this_thread::sleep_for")
```

### struct reset_agent

#### Public Functions

```cpp
reset_agent (detail::agent_storage *, agent_base &impl)
```

```cpp
reset_agent (agent_base &impl)
```

```cpp
~reset_agent ()
```

#### Public Members

```cpp
detail::agent_storage *storage_
```

```cpp
agent_base *old_
```

#include `<hpx/basic_execution/context_base.hpp>`

namespace hpx

namespace basic_execution

### struct context_base

#### Public Functions

```cpp
virtual ~context_base ()
```

```cpp
virtual resource_base const &resource () const = 0
```

#include `<hpx/basic_execution/agent_base.hpp>`

namespace hpx
namespace basic_execution

struct agent_base

Public Functions

virtual ~agent_base() 
virtual std::string description() const = 0
virtual context_base const &context() const = 0
virtual void yield(char const *desc) = 0
virtual void yield_k(std::size_t k, char const *desc) = 0
virtual void suspend(char const *desc) = 0
virtual void resume(char const *desc) = 0
virtual void abort(char const *desc) = 0
virtual void sleep_for(hpx::util::steady_duration const &sleep_duration, char const *desc) = 0
virtual void sleep_until(hpx::util::steady_time_point const &sleep_time, char const *desc) = 0

#include <hpx/basic_execution/register_locks.hpp>

namespace hpx

namespace util

Functions

bool register_lock(void const *, util::register_lock_data * = nullptr)
bool unregister_lock(void const *)
void verify_no_locks()
void force_error_on_lock()
void enable_lock_detection()
void ignore_lock(void const *)
void reset_ignored(void const *)
void ignore_all_locks()
void reset_ignored_all()
struct ignore_all_while_checking

Chapter 2. What’s so special about HPX?
Public Functions

ignore_all_while_checking()

template<
typename Lock,
typename Enable>
struct ignore_while_checking

Public Functions

ignore_while_checking(void const *)

#include <hpx/basic_execution/agent_ref.hpp>

namespace hpx

namespace basic_execution

class agent_ref

Public Functions

constexpr agent_ref()

constexpr agent_ref(agent_base *impl)

constexpr agent_ref(agent_ref const&)

HPX_CXX14_CONSTEXPR agent_ref& hpx::basic_execution::agent_ref::operator=(agent_ref const&)

HPX_CXX14_CONSTEXPR agent_ref& hpx::basic_execution::agent_ref::operator=(agent_ref&&)

constexpr operator bool() const

void reset(agent_base *impl = nullptr)

void yield(char const *desc = "hpx::basic_execution::agent_ref::yield")

void yield_k(std::size_t k, char const *desc = "hpx::basic_execution::agent_ref::yield_k")

void suspend(char const *desc = "hpx::basic_execution::agent_ref::suspend")

void resume(char const *desc = "hpx::basic_execution::agent_ref::resume")

void abort(char const *desc = "hpx::basic_execution::agent_ref::abort")

template<
typename Rep,
typename Period>
void sleep_for(std::chrono::duration<Rep, Period> const &sleep_duration, char const *desc = "hpx::basic_execution::agent_ref::sleep_for")

template<
typename Clock,
typename Duration>
void sleep_until(std::chrono::time_point<Clock, Duration> const &sleep_time, char const *desc = "hpx::basic_execution::agent_ref::sleep_until")

agent_base &ref()
Private Functions

void sleep_for (hpx::util::steady_duration const &sleep_duration, char const *desc)
void sleep_until (hpx::util::steady_time_point const &sleep_time, char const *desc)

Private Members

agent_base *impl_

Friends

cast<expr> bool operator== (agent_ref const &lhs, agent_ref const &rhs)
cast<expr> bool operator!= (agent_ref const &lhs, agent_ref const &rhs)
std::ostream &operator<< (std::ostream&, agent_ref const &)

#include <hpx/basic_execution/resource_base.hpp>

namespace hpx

namespace basic_execution

struct resource_base
#include <resource_base.hpp> TODO: implement, this is currently just a dummy.

Public Functions

virtual ~resource_base()
Functions

template <typename T = void>
std::string print_type(const char *delim = "")

template <>
std::string print_type(const char *)

template <typename T, typename... Args>
std::enable_if<sizeof...(Args) != 0, std::string>::type print_type(const char *delim = "")

template <typename T>
struct demangle_helper

Public Functions

cchar const *type_id() const

template <typename T>
struct type_id

Public Static Attributes

demangle_helper<T> typeid_ = demangle_helper<T>();

testing

#include <compatibility/hpx/util/lightweight_test.hpp>

#include <hpx/testing.hpp>

Defines

HPX_TEST (expr)
HPX_TEST_MSG (expr, msg)
HPX_TEST_EQ (expr1, expr2)
HPX_TEST_NEQ (expr1, expr2)
HPX_TEST_LT (expr1, expr2)
HPX_TEST_LTE (expr1, expr2)
HPX_TEST_RANGE (expr1, expr2, expr3)
HPX_TEST_EQ_MSG (expr1, expr2, msg)
HPX_TEST_NEQ_MSG (expr1, expr2, msg)
HPX_SANITY (expr)
HPX_SANITY_MSG (expr, msg)
HPX_SANITY_EQ (expr1, expr2)
HPX_SANITY_NEQ (expr1, expr2)
HPX_SANITY_LT (expr1, expr2)
HPX_SANITY_LTE (expr1, expr2)
HPX_SANITY_RANGE (expr1, expr2, expr3)
HPX_SANITY_EQ_MSG(expr1, expr2, msg)
HPX_TEST_THROW(expression, exception)

namespace hpx

namespace util

Typedefs

using hpx::util::test_failure_handler_type = typedef std::function<void()>

Enums

defined in [1.75]

enum counter_type
Values:

counter_sanity
counter_test

Functions

void set_test_failure_handler(test_failure_handler_type f)
int report_errors(std::ostream &stream = std::cerr)
void print_cdash_timing(const char *name, double time)
void print_cdash_timing(const char *name, std::uint64_t time)

functional

#include <compatibility/hpx/util/mem_fn.hpp>
#include <compatibility/hpx/util/protect.hpp>
#include <compatibility/hpx/util/invoke.hpp>
#include <compatibility/hpx/util/deferred_call.hpp>
#include <compatibility/hpx/util/bind_front.hpp>
#include <compatibility/hpx/util/invoke_fused.hpp>
#include <compatibility/hpx/util/first_argument.hpp>
#include <compatibility/hpx/util/function_ref.hpp>
#include <compatibility/hpx/util/bind.hpp>
#include <compatibility/hpx/util/bind_back.hpp>
#include <compatibility/hpx/util/result_of.hpp>
#include <compatibility/hpx/util/one_shot.hpp>
#include <compatibility/hpx/util/unique_function.hpp>
#include <hpx/functional/mem_fn.hpp>

namespace hpx

namespace util

Functions

template <typename M, typename C>
detail::mem_fn<M C::*>
mem_fn(M C::* pm)

template <typename R, typename C, typename... Ps>
detail::mem_fn<R (C::*)(Ps...)> hpx::util::mem_fn(R(C::*)(Ps...) pm)

template <typename R, typename C, typename... Ps>
detail::mem_fn<R (C::*)(Ps...) const> hpx::util::mem_fn(R(C::*)(Ps...) const pm)

#include <hpx/functional/protect.hpp>

namespace hpx

namespace util

Functions

template <typename T>
HPX_HOST_DEVICE std::enable_if< traits::is_bind_expression<typename std::decay<T>::type>::value, detail::protected_bind<typename std::decay<T>::type> >::type hpx::util::protect(T && f)

template <typename T>
HPX_HOST_DEVICE std::enable_if< !traits::is_bind_expression<typename std::decay<T>::type>::value, T>::type hpx::util::protect(T && v)

#include <hpx/functional/invoke.hpp>

Defines

HPX_INVOKE(F, ...)
HPX_INVOKE_R(R, F, ...)

namespace hpx

namespace util

Functions

template <typename F, typename... Ts>
HPX_HOST_DEVICE util::invoke_result<F, Ts...>::type hpx::util::invoke(F && f, Ts &&... vs)

Invokes the given callable object f with the content of the argument pack vs

Return  The result of the callable object when it’s called with the given argument types.
Note  This function is similar to std::invoke (C++17)
Parameters
• f: Requires to be a callable object. If f is a member function pointer, the first argument in the pack will be treated as the callee (this object).
• vs: An arbitrary pack of arguments

Exceptions
• std::exception: like objects thrown by call to object f with the argument types vs.

template <typename R, typename F, typename... Ts>
HPX_HOST_DEVICE R hpx::util::invoke_r(F && f, Ts &&... vs)  

• R: The result type of the function when it’s called with the content of the given argument types vs.

namespace functional

struct invoke

Public Functions

template <typename F, typename... Ts>
HPX_HOST_DEVICE util::invoke_result<F, Ts...>::type hpx::util::functional::invoke::operator()(F && f, Ts &&... vs) const

namespace hpx

namespace serialization

Functions

template <typename Archive, typename F, typename... Ts>
void serialize(Archive & ar, hpx::util::detail::deferred<F, Ts...> & d, unsigned int const version = 0)

namespace util

Functions

template <typename F, typename... Ts>
detail::deferred<typename std::decay<F>::type, typename std::decay<Ts>::type...> deferred_call(F && f, Ts &&... vs)

template <typename F>
std::decay<F>::type deferred_call(F && f)
```cpp
#include <hpx/functional/bind_front.hpp>

namespace hpx

namespace serialization

Functions

template <typename Archive, typename F, typename... Ts>
void serialize(Archive & ar, hpx::util::detail::bound_front<F, Ts...> & bounded, unsigned int const version = 0)

namespace util

Functions

template <typename F, typename... Ts>
detail::bound_front<typename std::decay<F>::type, typename std::decay<Ts>::type...> bind_front(F && f, Ts&&... vs)

template <typename F>
std::decay<F>::type bind_front(F && f)

#include <hpx/functional/invoke_fused.hpp>

namespace hpx

namespace util

Functions

template <typename F, typename Tuple>
HPX_HOST_DEVICE detail::invoke_fused_result<F, Tuple>::type hpx::util::invoke_fused(F && f, Tuple && t)
Invokes the given callable object f with the content of the sequenced type t (tuples, pairs)

Return The result of the callable object when it’s called with the content of the given sequenced type.

Note This function is similar to \texttt{std::apply} (C++17)

Parameters

\begin{itemize}
  \item \texttt{f}: Must be a callable object. If \texttt{f} is a member function pointer, the first argument in the sequenced type will be treated as the callee (this object).
  \item \texttt{t}: A type which is content accessible through a call to \texttt{hpx::util::get}.
\end{itemize}

Exceptions

\begin{itemize}
  \item \texttt{std::exception}: like objects thrown by call to object \texttt{f} with the arguments contained in the sequenceable type \texttt{t}.
\end{itemize}

template <typename R, typename F, typename Tuple>
HPX_HOST_DEVICE R hpx::util::invoke_fused_r(F && f, Tuple && t)

Template Parameters

\begin{itemize}
  \item \texttt{R}: The result type of the function when it’s called with the content of the given sequenced type.
\end{itemize}

#include <hpx/functional/first_argument.hpp>
```
```cpp
#include <hpx/functional/function_ref.hpp>

namespace hpx
{
    namespace util
    {
        template <typename R, typename... Ts>
        class function_ref<R (Ts...)>
        {
            // Public Functions
            template <typename F, typename FD = typename std::decay<F>::type, typename Enable = typename std::enable_if< !std::is_same<FD, function_ref>::value && traits::is_invocable_r<R, F&, Ts...>::value>::type>
            function_ref (function_ref const &other)
            {
            }
            template <typename F, typename FD = typename std::decay<F>::type, typename Enable = typename std::enable_if< !std::is_same<FD, function_ref>::value && traits::is_invocable_r<R, F&, Ts...>::value>::type>
            function_ref &operator=(F &&f)
            {
            }
            template <typename F, typename T = typename std::remove_reference<F>::type, typename Enable = typename std::enable_if<!std::is_pointer<T>::value>::type>
            void assign(F &&f)
            {
            }
            template<typename T>
            void assign(std::reference_wrapper<T> f_ref)
            {
            }
            template<typename T>
            void assign(T *f_ptr)
            {
            }
            void swap(function_ref &f)
            {
            }
            R operator()(Ts... vs) const
            {
            }
            std::size_t get_function_address() const
            {
            }
            char const *get_function_annotation() const
            {
            }
            util::itt::string_handle get_function_annotation_itt() const
            {
            }
            // Protected Attributes
            template<>
            R (*vptr)(void *, Ts&&...)
            {
            }
            void *object
            {
            }
            // Private Types
            template<>
            using VTable = detail::function_ref_vtable<R (Ts...)>
            {
            }
            // Private Static Functions
            template <typename T>
            static VTable const *get_vtable()
            {
            }
        }
    }
}
```
#include <hpx/functional/bind.hpp>

namespace hpx

namespace serialization

Functions

```cpp
template <typename Archive, typename F, typename... Ts>
void serialize(Archive &, hpx::util::detail::bound<F, Ts...> &bound, unsigned int const version = 0)
```

```cpp
template <typename Archive, std::size_t I>
void serialize(Archive &, hpx::util::detail::placeholder<I> &, unsigned int const = 0)
```

namespace util

Functions

```cpp
template <typename F, typename... Ts>
std::enable_if<!traits::is_action<typename std::decay<F>::type>::value, detail::bound<typename std::decay<F>::type, typename std::decay<Ts>::type...>>::type bind(F &&f, Ts&&... vs)
```

namespace placeholders

Variables

```cpp
HPX_STATIC_CONSTEXPR detail::placeholder<1> hpx::util::placeholders::_1 = {}
HPX_STATIC_CONSTEXPR detail::placeholder<2> hpx::util::placeholders::_2 = {}
HPX_STATIC_CONSTEXPR detail::placeholder<3> hpx::util::placeholders::_3 = {}
HPX_STATIC_CONSTEXPR detail::placeholder<4> hpx::util::placeholders::_4 = {}
HPX_STATIC_CONSTEXPR detail::placeholder<5> hpx::util::placeholders::_5 = {}
HPX_STATIC_CONSTEXPR detail::placeholder<6> hpx::util::placeholders::_6 = {}
HPX_STATIC_CONSTEXPR detail::placeholder<7> hpx::util::placeholders::_7 = {}
HPX_STATIC_CONSTEXPR detail::placeholder<8> hpx::util::placeholders::_8 = {}
HPX_STATIC_CONSTEXPR detail::placeholder<9> hpx::util::placeholders::_9 = {}
```

#include <hpx/functional/function.hpp>

Defines

```cpp
HPX_UTIL_REGISTER_FUNCTION_DECLARATION (Sig, F, Name)
HPX_UTIL_REGISTER_FUNCTION (Sig, F, Name)
```

namespace hpx

namespace util

2.9. API reference 395
**Typedefs**

```cpp
using hpx::util::function_nonser = typedef function<Sig, false>
```

```cpp
template <typename R, typename... Ts, bool Serializable>
template<class>

class function<R (Ts...), Serializable> : public detail::basic_function<R
Ts..., true, Serializable>
```

**Public Types**

```cpp
typedef R result_type
```

**Public Functions**

```cpp
function (std::nullptr_t = nullptr)
function (function const&)
function (function&&)
function &operator= (function const&)
function &operator= (function&&)
```

```cpp
template <typename F, typename FD = typename std::decay<F>::type, typename Enable1 = typename std::enable_if< !std::is_same<FD, function>::value>::type, typename Enable2 = typename std::enable_if< traits::is_invocable_r<R, FD&, Ts...>::value>::type>
function (F &&f)
```

```cpp
template <typename F, typename FD = typename std::decay<F>::type, typename Enable1 = typename std::enable_if< !std::is_same<FD, function>::value>::type, typename Enable2 = typename std::enable_if< traits::is_invocable_r<R, FD&, Ts...>::value>::type>
function &operator= (F &&f)
```

**Private Types**

```cpp
template<>
using base_type = detail::basic_function<R (Ts...), true, Serializable>
```

```cpp
#include <hpx/functional/bind_back.hpp>

namespace hpx

namespace serialization

Functions

template <typename Archive, typename F, typename... Ts>
void serialize(Archive &ar, hpx::util::detail::bound_back<F, Ts...> &bound, unsigned int const version = 0)
```

```cpp
namespace util

Functions

template <typename F, typename... Ts>
detail::bound_back<typename std::decay<F>::type, typename std::decay<Ts>::type...> bind_back (F &&f, Ts&&... vs)
```

396 Chapter 2. What’s so special about HPX?
template <typename F>
std::decay<F>::type bind_back (F &&f)

#include <hpx/functional/result_of.hpp>
#include <hpx/functional/one_shot.hpp>

namespace hpx

namespace serialization

Functions

template <typename Archive, typename F>
void serialize(Archive &ar, hpx::util::detail::one_shot_wrapper<F> &one_shot_wrapper, unsigned int const version = 0)

namespace util

Functions

template <typename F>
detail::one_shot_wrapper<typename std::decay<F>::type> one_shot (F &&f)

#include <hpx/functional/unique_function.hpp>

Defines

HPX_UTIL_REGISTER_UNIQUE_FUNCTION_DECLARATION (Sig, F, Name)
HPX_UTIL_REGISTER_UNIQUE_FUNCTION (Sig, F, Name)

namespace hpx

namespace util

Typedefs

using hpx::util::unique_function_nonser = typedef unique_function<Sig, false>

template <typename R, typename... Ts, bool Serializable>
template<class>

class unique_function<R (Ts...), Serializable> : public detail::basic_function<R Ts..., false, Serializable>

Public Types

typedef R result_type

Public Functions

unique_function (std::nullptr_t = nullptr)
unique_function (unique_function&&)
unique_function &operator=(unique_function&&)

```cpp
template <typename F, typename FD = typename std::decay<F>::type, typename Enable1 = typename std::enable_if< !std::is_same<FD, unique_function>::value>::type,
            typename Enable2 = typename std::enable_if< traits::is_invocable_r<R, FD&, Ts...>::value>::type>
unique_function
```

```cpp
template <typename F && f>
```

### Private Types

```cpp
template<>
using base_type = detail::basic_function<R (Ts...), false, Serializable>
```

```cpp
#include <hpx/functional/traits/is_bind_expression.hpp>
```

```cpp
namespace hpx
```

```cpp
namespace traits
```

```cpp
template <typename T>
struct is_bind_expression : public false_type
```

```cpp
Include hpx::traits::is_bind_expression<T const>
```

```cpp
#include <hpx/functional/traits/get_function_annotation.hpp>
```

```cpp
namespace hpx
```

```cpp
namespace traits
```

```cpp
template <typename F, typename Enable = void>
struct get_function_annotation
```

### Public Static Functions

```cpp
static char const *call(F const &)
```

```cpp
#include <hpx/functional/traits/get_function_address.hpp>
```

```cpp
template <typename R, typename Obj, typename... Ts>
template<>
struct get_function_address<R (Obj::*)(Ts...)>
```

```cpp
Public Static Functions
```

```cpp
static std::size_t hpx::traits::get_function_address::call(R(Obj::*)(Ts...) f)
```

```cpp
template <typename R, typename Obj, typename... Ts>
template<>
struct get_function_address<R (Obj::*)(Ts...) const>
```

```cpp
Public Static Functions
```

```cpp
static std::size_t hpx::traits::get_function_address::call(R(Obj::*)(Ts...) f)
```
Public Static Functions

static std::size_t hpx::traits::get_function_address::call(R(Obj::*)(Ts...) const f)

namespace hpx

namespace traits

    template <typename F, typename Enable = void>
    struct get_function_address

    Public Static Functions

        static std::size_t call (F const &f)

    template <typename R, typename... Ts>
    template<>
    struct get_function_address<R (*)(Ts...)>

    Public Static Functions

        static std::size_t call (R (*)(Ts...) Ts...)

    template <typename R, typename Obj, typename... Ts>
    template<>
    struct get_function_address<R (Obj::*)(Ts...) const>

    Public Static Functions

        static std::size_t hpx::traits::get_function_address::call(R(Obj::*)(Ts...) const f)

#include <hpx/functional/traits/is_action.hpp>
#include <hpx/functional/traits/is_callable.hpp>
#include <hpx/functional/traits/is_placeholder.hpp>

namespace hpx

namespace traits

    template <typename T>
    struct is_placeholder : public boost::is_placeholder<T>

        Subclassed by hpx::traits::is_placeholder<T const>
hardware
#include <compatibility/hpx/util/hardware/timestamp.hpp>
#include <compatibility/hpx/util/hardware/bit_manipulation.hpp>
#include <compatibility/hpx/util/hardware/cpuid.hpp>
#include <compatibility/hpx/util/hardware/timestamp/linux_x86_64.hpp>
#include <compatibility/hpx/util/hardware/timestamp/linux_x86_32.hpp>
#include <compatibility/hpx/util/hardware/timestamp/msvc.hpp>
#include <compatibility/hpx/util/hardware/timestamp/linux_generic.hpp>
#include <compatibility/hpx/util/hardware/timestamp/bgq.hpp>
#include <compatibility/hpx/util/hardware/cpuid/linux_x86.hpp>
#include <compatibility/hpx/util/hardware/cpuid/msvc.hpp>
#include <hpx/hardware/timestamp.hpp>
#include <hpx/hardware/bit_manipulation.hpp>
namespace hpx
    namespace util
        namespace hardware

Functions

template <typename T, typename U>
bool has_bit_set(T value, U bit)

template <std::size_t N, typename T>
T unbounded_shl(T x)

template <std::size_t N, typename T>
T unbounded_shr(T x)

template <std::size_t Low, std::size_t High, typename Result, typename T>
Result get_bit_range(T x)

template <std::size_t Low, typename Result, typename T>
Result pack_bits(T x)

template <std::size_t N, typename T>
struct unbounded_shifter

Public Static Functions

static T shl(T x)

static T shr(T x)
template <typename T>
template<>
struct unbounded_shifter<0, T>

Public Static Functions

static T shl(T x)
static T shr(T x)

#include <hpx/hardware/cpuid.hpp>
#include <hpx/hardware/timestamp/linux_x86_64.hpp>
namespace hpx

namespace util

namespace hardware

Functions

std::uint64_t timestamp()

#include <hpx/hardware/timestamp/linux_x86_32.hpp>
#include <hpx/hardware/timestamp/msvc.hpp>
#include <hpx/hardware/timestamp/linux_generic.hpp>
#include <hpx/hardware/timestamp/bgq.hpp>
#include <hpx/hardware/cpuid/linux_x86.hpp>
namespace hpx

namespace util

namespace hardware

Functions

void cpuid(std::uint32_t (&cpuinfo)[4], std::uint32_t eax)
void cpuidex(std::uint32_t (&cpuinfo)[4], std::uint32_t eax, std::uint32_t ecx)

struct cpuid_register

Public Types

enum info
Values:
eax = 0
ebx = 1
cx = 2
dx = 3

#include <hpx/hardware/cpuid/msvc.hpp>
collectives
#include <compatibility/hpx/lcos/reduce.hpp>
#include <compatibility/hpx/lcos/gather.hpp>
#include <compatibility/hpx/lcos/broadcast.hpp>
#include <compatibility/hpx/lcos/latch.hpp>
#include <compatibility/hpx/lcos/barrier.hpp>
#include <compatibility/hpx/lcos/fold.hpp>
#include <compatibility/hpx/lcos/spmd_block.hpp>
#include <hpx/collectives/reduce.hpp>

namespace hpx

namespace lcos

Functions

template <typename Action, typename ReduceOp, typename ArgN, ...>
hpx::future<decltype(Action(hpx::id_type, ArgN, ...))> reduce(decltype(Action(hpx::id_type, ArgN, ...))) reduce(std::vector<hpx::id_type> const &ids, ReduceOp &&reduce_op, ArgN argN, ...)

Perform a distributed reduction operation.

The function hpx::lcos::reduce performs a distributed reduction operation over results returned from action invocations on a given set of global identifiers. The action can be either a plain action (in which case the global identifiers have to refer to localities) or a component action (in which case the global identifiers have to refer to instances of a component type which exposes the action.

Return This function returns a future representing the result of the overall reduction operation.

Parameters

• ids: [in] A list of global identifiers identifying the target objects for which the given action will be invoked.
• reduce_op: [in] A binary function expecting two results as returned from the action invocations. The function (or function object) is expected to return the result of the reduction operation performed on its arguments.
• argN: [in] Any number of arbitrary arguments (passed by by const reference) which will be forwarded to the action invocation.
Perform a distributed reduction operation.

The function `hpx::lcos::reduce_with_index` performs a distributed reduction operation over results returned from action invocations on a given set of global identifiers. The action can be either plain action (in which case the global identifiers have to refer to localities) or a component action (in which case the global identifiers have to refer to instances of a component type which exposes the action.

The function passes the index of the global identifier in the given list of identifiers as the last argument to the action.

**Return** This function returns a future representing the result of the overall reduction operation.

**Parameters**
- `ids`: [in] A list of global identifiers identifying the target objects for which the given action will be invoked.
- `reduce_op`: [in] A binary function expecting two results as returned from the action invocations. The function (or function object) is expected to return the result of the reduction operation performed on its arguments.
- `argN`: [in] Any number of arbitrary arguments (passed by by const reference) which will be forwarded to the action invocation.

```
#include <hpx/collectives/gather.hpp>
```

**Defines**

```
HPX_REGISTER_GATHER_DECLARATION (type, name)
```

Declare a gather object named `name` for a given data type `type`.

The macro `HPX_REGISTER_GATHER_DECLARATION` can be used to declare all facilities necessary for a (possibly remote) gather operation.

The parameter `type` specifies for which data type the gather operations should be enabled.

The (optional) parameter `name` should be a unique C-style identifier which will be internally used to identify a particular gather operation. If this defaults to `<type>_gather` if not specified.

**Note** The macro `HPX_REGISTER_GATHER_DECLARATION` can be used with 1 or 2 arguments. The second argument is optional and defaults to `<type>_gather`.

```
HPX_REGISTER_GATHER (type, name)
```

Define a gather object named `name` for a given data type `type`.

The macro `HPX_REGISTER_GATHER` can be used to define all facilities necessary for a (possibly remote) gather operation.

The parameter `type` specifies for which data type the gather operations should be enabled.

The (optional) parameter `name` should be a unique C-style identifier which will be internally used to identify a particular gather operation. If this defaults to `<type>_gather` if not specified.
Note The macro `HPX_REGISTER_GATHER` can be used with 1 or 2 arguments. The second argument is optional and defaults to `<type>_gather`.

```cpp
namespace hpx
{
namespace lcos
{

Functions

template <typename T>
hpx::future<std::vector<T>> gather_here(char const *basename, hpx::future<T> result,
std::size_t num_sites = std::size_t(-1), std::size_t
generation = std::size_t(-1), std::size_t this_site =
std::size_t(-1))

Gather a set of values from different call sites

This function receives a set of values from all call sites operating on the given base name.

Note Each gather operation has to be accompanied with a unique usage of the `HPX_REGISTER_GATHER` macro to define the necessary internal facilities used by `gather_here` and `gather_there`

Return This function returns a future holding a vector with all gathered values. It will become ready once the gather operation has been completed.

Parameters
• basename: The base name identifying the gather operation
• result: A future referring to the value to transmit to the central gather point from this call site.
• num_sites: The number of participating sites (default: all localities).
• generation: The generational counter identifying the sequence number of the gather operation performed on the given base name. This is optional and needs to be supplied only if the gather operation on the given base name has to be performed more than once.
• this_site: The sequence number of this invocation (usually the locality id). This value is optional and defaults to whatever hpx::get_locality_id() returns.

template <typename T>
hpx::future<void> gather_there(char const *basename, hpx::future<T> result,
std::size_t
generation = std::size_t(-1), std::size_t root_site = 0, std::size_t
this_site = std::size_t(-1))

Gather a given value at the given call site

This function transmits the value given by `result` to a central gather site (where the corresponding `gather_here` is executed)

Note Each gather operation has to be accompanied with a unique usage of the `HPX_REGISTER_GATHER` macro to define the necessary internal facilities used by `gather_here` and `gather_there`

Return This function returns a future which will become ready once the gather operation has been completed.

Parameters
• basename: The base name identifying the gather operation
• result: A future referring to the value to transmit to the central gather point from this call site.
• generation: The generational counter identifying the sequence number of the gather operation performed on the given base name. This is optional and needs to be supplied only if the gather operation on the given base name has to be performed more than once.
```
• **root_site**: The sequence number of the central gather point (usually the locality id). This value is optional and defaults to 0.
• **this_site**: The sequence number of this invocation (usually the locality id). This value is optional and defaults to whatever `hpx::get_locality_id()` returns.

```cpp
template <typename T>
hpx::future<std::vector<T>> gather_here(char const *basename, T &&result, std::size_t num_sites = std::size_t(-1), std::size_t generation = std::size_t(-1), std::size_t this_site = std::size_t(-1))
```

Gather a set of values from different call sites

This function receives a set of values from all call sites operating on the given base name.

**Note** Each gather operation has to be accompanied with a unique usage of the `HPX_REGISTER_GATHER` macro to define the necessary internal facilities used by `gather_here` and `gather_there`

**Return** This function returns a future holding a vector with all gathered values. It will become ready once the gather operation has been completed.

**Parameters**

- **basename**: The base name identifying the gather operation
- **result**: The value to transmit to the central gather point from this call site.
- **num_sites**: The number of participating sites (default: all localities).
- **generation**: The generational counter identifying the sequence number of the gather operation performed on the given base name. This is optional and needs to be supplied only if the gather operation on the given base name has to be performed more than once.
- **this_site**: The sequence number of this invocation (usually the locality id). This value is optional and defaults to whatever `hpx::get_locality_id()` returns.

```cpp
template <typename T>
hpx::future<void> gather_there(char const *basename, T &&result, std::size_t generation = std::size_t(-1), std::size_t root_site = 0, std::size_t this_site = std::size_t(-1))
```

Gather a given value at the given call site

This function transmits the value given by `result` to a central gather site (where the corresponding `gather_here` is executed)

**Note** Each gather operation has to be accompanied with a unique usage of the `HPX_REGISTER_GATHER` macro to define the necessary internal facilities used by `gather_here` and `gather_there`

**Return** This function returns a future which will become ready once the gather operation has been completed.

**Parameters**

- **basename**: The base name identifying the gather operation
- **result**: The value to transmit to the central gather point from this call site.
- **generation**: The generational counter identifying the sequence number of the gather operation performed on the given base name. This is optional and needs to be supplied only if the gather operation on the given base name has to be performed more than once.
- **root_site**: The sequence number of the central gather point (usually the locality id). This value is optional and defaults to 0.
- **this_site**: The sequence number of this invocation (usually the locality id). This value is optional and defaults to whatever `hpx::get_locality_id()` returns.
#include <hpx/collectives/all_reduce.hpp>

Defines

**HPX_REGISTER_ALLREDUCE_DECLARATION** (type, name)

Declare a `all_reduce` object named `name` for a given data type `type`.

The macro `HPX_REGISTER_ALLREDUCE_DECLARATION` can be used to declare all facilities necessary for a (possibly remote) `all_reduce` operation.

The parameter `type` specifies for which data type the `all_reduce` operations should be enabled.

The (optional) parameter `name` should be a unique C-style identifier that will be internally used to identify a particular `all_reduce` operation. If this defaults to `<type>_all_reduce` if not specified.

**Note** The macro `HPX_REGISTER_ALLREDUCE_DECLARATION` can be used with 1 or 2 arguments. The second argument is optional and defaults to `<type>_all_reduce`.

**HPX_REGISTER_ALLREDUCE** (type, name)

Define a `all_reduce` object named `name` for a given data type `type`.

The macro `HPX_REGISTER_ALLREDUCE` can be used to define all facilities necessary for a (possibly remote) `all_reduce` operation.

The parameter `type` specifies for which data type the `all_reduce` operations should be enabled.

The (optional) parameter `name` should be a unique C-style identifier that will be internally used to identify a particular `all_reduce` operation. If this defaults to `<type>_all_reduce` if not specified.

**Note** The macro `HPX_REGISTER_ALLREDUCE` can be used with 1 or 2 arguments. The second argument is optional and defaults to `<type>_all_reduce`.

namespace hpx

namespace lcos

Functions

```cpp
template <typename T, typename F>
hpx::future<T> all_reduce (char const *basename, hpx::future<T> result, F &&op, std::size_t num_sites = std::size_t(-1), std::size_t generation = std::size_t(-1), std::size_t this_site = std::size_t(-1), std::size_t root_site = 0)
```

AllReduce a set of values from different call sites.

This function receives a set of values that are the result of applying a given operator on values supplied from all call sites operating on the given base name.

**Note** Each `all_reduce` operation has to be accompanied with a unique usage of the `HPX_REGISTER_ALLREDUCE` macro to define the necessary internal facilities used by `all_reduce`.

**Return** This function returns a future holding a vector with all values send by all participating sites. It will become ready once the `all_reduce` operation has been completed.

**Parameters**

- `basename`: The base name identifying the `all_reduce` operation
- `local_result`: A future referring to the value to transmit to all participating sites from this call site.
- `op`: Reduction operation to apply to all values supplied from all participating sites
- `num_sites`: The number of participating sites (default: all localities).
• **generation**: The generational counter identifying the sequence number of the all_reduce operation performed on the given base name. This is optional and needs to be supplied only if the all_reduce operation on the given base name has to be performed more than once.

• **this_site**: The sequence number of this invocation (usually the locality id). This value is optional and defaults to whatever hpx::get_locality_id() returns. root_site The site that is responsible for creating the all_reduce support object. This value is optional and defaults to ‘0’ (zero).

```cpp
template <typename T, typename F>
hpx::future<std::decay_t<T>> all_reduce(char const *basename, T &&result, F &&op,
std::size_t num_sites = std::size_t(-1), std::size_t
generation = std::size_t(-1), std::size_t this_site =
std::size_t(-1), std::size_t root_site = 0)
```

AllReduce a set of values from different call sites

This function receives a set of values from all call sites operating on the given base name.

**Note** Each all_reduce operation has to be accompanied with a unique usage of the `HPX_REGISTER_ALLREDUCE` macro to define the necessary internal facilities used by all_reduce.

**Return** This function returns a future holding a vector with all values send by all participating sites. It will become ready once the all_reduce operation has been completed.

**Parameters**

- **basename**: The base name identifying the all_reduce operation
- **local_result**: The value to transmit to all participating sites from this call site.
- **op**: Reduction operation to apply to all values supplied from all participating sites
- **num_sites**: The number of participating sites (default: all localities).
- **generation**: The generational counter identifying the sequence number of the all_reduce operation performed on the given base name. This is optional and needs to be supplied only if the all_reduce operation on the given base name has to be performed more than once.
- **this_site**: The sequence number of this invocation (usually the locality id). This value is optional and defaults to whatever hpx::get_locality_id() returns. root_site The site that is responsible for creating the all_reduce support object. This value is optional and defaults to ‘0’ (zero).

```cpp
#include <hpx/collectives/broadcast.hpp>
namespace hpx

namespace lcos

Functions

```cpp
template <typename Action, typename ArgN, ...>
hpx::future<std::vector<decltype(Action(hpx::id_type, ArgN, ...))>> broadcast(std::vector<hpx::id_type>
const &kids,
ArgN argN, ...)
```

Perform a distributed broadcast operation.

The function hpx::lcos::broadcast performs a distributed broadcast operation resulting in action invocations on a given set of global identifiers. The action can be either a plain action (in which case the global identifiers have to refer to localities) or a component action (in which case the global identifiers have to refer to instances of a component type which exposes the action.

The given action is invoked asynchronously on all given identifiers, and the arguments ArgN are passed along to those invocations.
**Return** This function returns a future representing the result of the overall reduction operation.

**Note** If `decltype(Action(...))` is void, then the result of this function is `future<void>`.

**Parameters**
- `ids`: [in] A list of global identifiers identifying the target objects for which the given action will be invoked.
- `argN`: [in] Any number of arbitrary arguments (passed by const reference) which will be forwarded to the action invocation.

```cpp
template <typename Action, typename ArgN, ...>
void broadcast_apply (std::vector<hpx::id_type> const &ids, ArgN argN, ...)
```

Perform an asynchronous (fire&forget) distributed broadcast operation.

The function `hpx::lcos::broadcast_apply` performs an asynchronous (fire&forget) distributed broadcast operation resulting in action invocations on a given set of global identifiers. The action can be either a plain action (in which case the global identifiers have to refer to localities) or a component action (in which case the global identifiers have to refer to instances of a component type which exposes the action.

The given action is invoked asynchronously on all given identifiers, and the arguments `ArgN` are passed along to those invocations.

**Parameters**
- `ids`: [in] A list of global identifiers identifying the target objects for which the given action will be invoked.
- `argN`: [in] Any number of arbitrary arguments (passed by const reference) which will be forwarded to the action invocation.

```cpp
template <typename Action, typename ArgN, ...>
hpx::future<std::vector<decltype(Action(hpx::id_type, ArgN, ..., std::size_t))>> broadcast_with_index (std::vector<hpx::id_type> const &ids, ArgN argN, ...)
```

Perform a distributed broadcast operation.

The function `hpx::lcos::broadcast_with_index` performs a distributed broadcast operation resulting in action invocations on a given set of global identifiers. The action can be either a plain action (in which case the global identifiers have to refer to localities) or a component action (in which case the global identifiers have to refer to instances of a component type which exposes the action.

The given action is invoked asynchronously on all given identifiers, and the arguments `ArgN` are passed along to those invocations.

The function passes the index of the global identifier in the given list of identifiers as the last argument to the action.

**Return** This function returns a future representing the result of the overall reduction operation.

**Note** If `decltype(Action(...))` is void, then the result of this function is `future<void>`.

**Parameters**
- `ids`: [in] A list of global identifiers identifying the target objects for which the given action will be invoked.
- `argN`: [in] Any number of arbitrary arguments (passed by const reference) which will be forwarded to the action invocation.

```cpp
template <typename Action, typename ArgN, ...>
void broadcast_apply_with_index (std::vector<hpx::id_type> const &ids, ArgN argN, ...)
```

Perform an asynchronous (fire&forget) distributed broadcast operation.
The function `hpx::lcos::broadcast_apply_with_index` performs an asynchronous (fire&forget) distributed broadcast operation resulting in action invocations on a given set of global identifiers. The action can be either a plain action (in which case the global identifiers have to refer to localities) or a component action (in which case the global identifiers have to refer to instances of a component type which exposes the action.

The given action is invoked asynchronously on all given identifiers, and the arguments `ArgN` are passed along to those invocations.

The function passes the index of the global identifier in the given list of identifiers as the last argument to the action.

**Parameters**

- `ids`: [in] A list of global identifiers identifying the target objects for which the given action will be invoked.
- `argN`: [in] Any number of arbitrary arguments (passed by const reference) which will be forwarded to the action invocation.

```cpp
#include <hpx/collectives/latch.hpp>

namespace hpx

namespace lcos

class latch : public components::client_base<latch, lcos::server::latch>

Public Functions

latch()
latch(std::ptrdiff_t count)
    Initialize the latch
    Requires: count >= 0. Synchronization: None Postconditions: counter_ == count.
latch(naming::id_type const &id)
    Extension: Create a client side representation for the existing server::latch instance with the given global id id.
latch(hpx::future<naming::id_type> &&f)
    Extension: Create a client side representation for the existing server::latch instance with the given global id id.
latch(hpx::shared_future<naming::id_type> const &id)
    Extension: Create a client side representation for the existing server::latch instance with the given global id id.
latch(hpx::shared_future<naming::id_type> &&id)
    void count_down_and_wait()
        Decrement counter_ by 1. Blocks at the synchronization point until counter_ reaches 0.
        Requires: counter_ > 0.
        Synchronization: Synchronizes with all calls that block on this latch and with all is_ready calls on this latch that return true.
```
Exceptions

- Nothing.

```cpp
void count_down(std::ptrdiff_t n)
Decrements counter_ by n. Does not block.

Requires: counter_ >= n and n >= 0.
Synchronization: Synchronizes with all calls that block on this latch and with all is_ready calls on this latch that return true.
```

Exceptions

- Nothing.

```cpp
bool is_ready() const
Returns: counter_ == 0. Does not block.
```

Exceptions

- Nothing.

```cpp
void wait() const
If counter_ is 0, returns immediately. Otherwise, blocks the calling thread at the synchronization point until counter_ reaches 0.
```

Exceptions

- Nothing.

Private Types

```cpp
typedef components::client_base<latch, lcos::server::latch> base_type
```

```cpp
#include <hpx/collectives/barrier.hpp>
namespace hpx
```

namespace lcos

```cpp
class barrier
#include <barrier.hpp> The barrier is an implementation performing a barrier over a number of participating threads. The different threads don’t have to be on the same locality. This barrier can be invoked in a distributed application.

For a local only barrier
See hpx::lcos::local::barrier.
```

Public Functions

```cpp
barrier(std::string const &base_name)
Creates a barrier, rank is locality id, size is number of localities

A barrier base_name is created. It expects that hpx::get_num_localities() participate and the local rank is hpx::get_locality_id().
```

Parameters

- base_name: The name of the barrier
barrier (std::string const &base_name, std::size_t num)
  Creates a barrier with a given size, rank is locality id
  A barrier base_name is created. It expects that num participate and the local rank is hpx::get_locality_id().
  Parameters
  • base_name: The name of the barrier
  • num: The number of participating threads

barrier (std::string const &base_name, std::size_t num, std::size_t rank)
  Creates a barrier with a given size and rank
  A barrier base_name is created. It expects that num participate and the local rank is rank.
  Parameters
  • base_name: The name of the barrier
  • num: The number of participating threads
  • rank: The rank of the calling site for this invocation

barrier (std::string const &base_name, std::vector<std::size_t> const &ranks, std::size_t rank)
  Creates a barrier with a vector of ranks
  A barrier base_name is created. It expects that ranks.size() and the local rank is rank (must be contained in ranks).
  Parameters
  • base_name: The name of the barrier
  • ranks: Gives a list of participating ranks (this could be derived from a list of locality ids
  • rank: The rank of the calling site for this invocation

void wait ()
  Wait until each participant entered the barrier. Must be called by all participants
  Return This function returns once all participants have entered the barrier (have called wait).

hpx::future<void> wait (hpx::launch::async_policy)
  Wait until each participant entered the barrier. Must be called by all participants
  Return a future that becomes ready once all participants have entered the barrier (have called wait).

Public Static Functions

static void synchronize ()
  Perform a global synchronization using the default global barrier The barrier is created once at startup and can be reused throughout the lifetime of an HPX application.
  Note This function currently does not support dynamic connection and disconnection of locali-

#include <hpx/collectives/all_to_all.hpp>

Defines

HPX_REGISTER_ALLTOALL_DECLARATION (type, name)
  Declare a all_to_all object named name for a given data type type.
  The macro HPX_REGISTER_ALLTOALL DECLARATION can be used to declare all facilities necessary for a (possibly remote) all_to_all operation.
The parameter `type` specifies for which data type the `all_to_all` operations should be enabled.

The (optional) parameter `name` should be a unique C-style identifier that will be internally used to identify a particular `all_to_all` operation. If this defaults to `<type>_all_to_all` if not specified.

**Note** The macro `HPX_REGISTER_ALLTOALL DECLARATION` can be used with 1 or 2 arguments. The second argument is optional and defaults to `<type>_all_to_all`.

```
HPX_REGISTER_ALLTOALL (type, name)
```

Define a all_to_all object named `name` for a given data type `type`.

The macro `HPX_REGISTER_ALLTOALL` can be used to define all facilities necessary for a (possibly remote) all_to_all operation.

The parameter `type` specifies for which data type the all_to_all operations should be enabled.

The (optional) parameter `name` should be a unique C-style identifier that will be internally used to identify a particular all_to_all operation. If this defaults to `<type>_all_to_all` if not specified.

**Note** The macro `HPX_REGISTER_ALLTOALL` can be used with 1 or 2 arguments. The second argument is optional and defaults to `<type>_all_to_all`.

```
namespace hpx

namespace lcos

Functions

template <typename T>
hpx::future<std::vector<T>> all_to_all (char const *basename, hpx::future<T> result,
          std::size_t num_sites = std::size_t(-1), std::size_t
generation = std::size_t(-1), std::size_t this_site =
          std::size_t(-1), std::size_t root_site = 0)

AllToAll a set of values from different call sites
This function receives a set of values from all call sites operating on the given base name.

**Note** Each all_to_all operation has to be accompanied with a unique usage of the `HPX_REGISTER_ALLTOALL` macro to define the necessary internal facilities used by all_to_all.

**Return** This function returns a future holding a vector with all values send by all participating sites. It will become ready once the all_to_all operation has been completed.

**Parameters**

- `basename`: The base name identifying the all_to_all operation
- `local_result`: A future referring to the value to transmit to all participating sites from this call site.
- `num_sites`: The number of participating sites (default: all localities).
- `generation`: The generational counter identifying the sequence number of the all_to_all operation performed on the given base name. This is optional and needs to be supplied only if the all_to_all operation on the given base name has to be performed more than once.
- `this_site`: The sequence number of this invocation (usually the locality id). This value is optional and defaults to whatever `hpx::get_locality_id()` returns. `root_site` The site that is responsible for creating the all_to_all support object. This value is optional and defaults to ‘0’ (zero).

```
```
AllToAll a set of values from different call sites

This function receives a set of values from all call sites operating on the given base name.

**Note** Each all_to_all operation has to be accompanied with a unique usage of the `HPX_REGISTER_ALLTOALL` macro to define the necessary internal facilities used by `all_to_all`.

**Return** This function returns a future holding a vector with all values send by all participating sites. It will become ready once the all_to_all operation has been completed.

**Parameters**
- `basename`: The base name identifying the all_to_all operation
- `local_result`: The value to transmit to all participating sites from this call site.
- `num_sites`: The number of participating sites (default: all localities).
- `generation`: The generational counter identifying the sequence number of the all_to_all operation performed on the given base name. This is optional and needs to be supplied only if the all_to_all operation on the given base name has to be performed more than once.
- `this_site`: The sequence number of this invocation (usually the locality id). This value is optional and defaults to whatever `hpx::get_locality_id()` returns. `root_site` The site that is responsible for creating the all_to_all support object. This value is optional and defaults to `0` (zero).

```cpp
#include <hpx/collectives/fold.hpp>

namespace hpx
{
    namespace lcos
    {

    Functions

    template <typename Action, typename FoldOp, typename Init, typename ArgN, ...>
    hpx::future<decltype(Action(hpx::id_type, ArgN, ...))> fold (std::vector<hpx::id_type> const &ids, FoldOp &&&fold_op, Init &&&init, ArgN argN, ...)

    Perform a distributed fold operation.

    The function `hpx::lcos::fold` performs a distributed folding operation over results returned from action invocations on a given set of global identifiers. The action can be either a plain action (in which case the global identifiers have to refer to localities) or a component action (in which case the global identifiers have to refer to instances of a component type which exposes the action).

    **Note** The type of the initial value must be convertible to the result type returned from the invoked action.

    **Return** This function returns a future representing the result of the overall folding operation.

    **Parameters**
    - `ids`: [in] A list of global identifiers identifying the target objects for which the given action will be invoked.
• **fold_op**: [in] A binary function expecting two results as returned from the action invocations. The function (or function object) is expected to return the result of the folding operation performed on its arguments.
• **init**: [in] The initial value to be used for the folding operation
• **argN**: [in] Any number of arbitrary arguments (passed by value, by const reference or by rvalue reference) which will be forwarded to the action invocation.

```cpp
template<typename Action, typename FoldOp, typename Init, typename ArgN, ...>
hpx::future<decltype(Action(hpx::id_type, ArgN, ..., std::size_t))>
fold_with_index(const std::vector<hpx::id_type>& ids, FoldOp &&fold_op, Init &&init, ArgN argN, ...)
```

Perform a distributed folding operation.

The function `hpx::lcos::fold_with_index` performs a distributed folding operation over results returned from action invocations on a given set of global identifiers. The action can be either plain action (in which case the global identifiers have to refer to localities) or a component action (in which case the global identifiers have to refer to instances of a component type which exposes the action.

The function passes the index of the global identifier in the given list of identifiers as the last argument to the action.

**Note** The type of the initial value must be convertible to the result type returned from the invoked action.

**Return** This function returns a future representing the result of the overall folding operation.

**Parameters**
• **ids**: [in] A list of global identifiers identifying the target objects for which the given action will be invoked.
• **fold_op**: [in] A binary function expecting two results as returned from the action invocations. The function (or function object) is expected to return the result of the folding operation performed on its arguments.
• **init**: [in] The initial value to be used for the folding operation
• **argN**: [in] Any number of arbitrary arguments (passed by value, by const reference or by rvalue reference) which will be forwarded to the action invocation.

```cpp
template<typename Action, typename FoldOp, typename Init, typename ArgN, ...>
hpx::future<decltype(Action(hpx::id_type, ArgN, ...))>
inverse_fold(const std::vector<hpx::id_type>& ids, FoldOp &&fold_op, Init &&init, ArgN argN, ...)
```

Perform a distributed inverse folding operation.

The function `hpx::lcos::inverse_fold` performs an inverse distributed folding operation over results returned from action invocations on a given set of global identifiers. The action can be either a plain action (in which case the global identifiers have to refer to localities) or a component action (in which case the global identifiers have to refer to instances of a component type which exposes the action.

**Note** The type of the initial value must be convertible to the result type returned from the invoked action.

**Return** This function returns a future representing the result of the overall folding operation.

**Parameters**
• **ids**: [in] A list of global identifiers identifying the target objects for which the given action will be invoked.
• **fold_op**: [in] A binary function expecting two results as returned from the action invocations. The function (or function object) is expected to return the result of the folding operation performed on its arguments.
• **init**: [in] The initial value to be used for the folding operation
• **argN**: [in] Any number of arbitrary arguments (passed by value, by const reference or by rvalue reference) which will be forwarded to the action invocation.

```cpp
template <typename Action, typename FoldOp, typename Init, typename ArgN,...>
hpx::future<decltype(Action(hpx::id_type, ArgN, ..., std::size_t))> inverse_fold_with_index
    (std::vector<hpx::id_type> const& ids, FoldOp &&fold_op, Init &&init, ArgN argN, ...
```

Perform a distributed inverse folding operation.

The function `hpx::lcos::inverse_fold_with_index` performs an inverse distributed folding operation over results returned from action invocations on a given set of global identifiers. The action can be either plain action (in which case the global identifiers have to refer to localities) or a component action (in which case the global identifiers have to refer to instances of a component type which exposes the action.

The function passes the index of the global identifier in the given list of identifiers as the last argument to the action.

**Note** The type of the initial value must be convertible to the result type returned from the invoked action.

**Return** This function returns a future representing the result of the overall folding operation.

**Parameters**
• **ids**: [in] A list of global identifiers identifying the target objects for which the given action will be invoked.
• **fold_op**: [in] A binary function expecting two results as returned from the action invocations. The function (or function object) is expected to return the result of the folding operation performed on its arguments.
• **init**: [in] The initial value to be used for the folding operation
• **argN**: [in] Any number of arbitrary arguments (passed by value, by const reference or by rvalue reference) which will be forwarded to the action invocation.

```cpp
#include <hpx/collectives/spmd_block.hpp>
namespace hpx

namespace lcos

Functions

```cpp
template <typename F, typename... Args>
hpx::future<void> define_spmd_block (std::string &&name, std::size_t images_per_locality, F &&f, Args&&... args)
```

The class `spmd_block` defines an interface for launching multiple images
while giving handles to each image to interact with the remaining images. The `define_spmd_block` function templates create multiple images of a user-defined action and launches them in a possibly separate thread. A temporary `spmd_block` object is created and diffused to each image. The constraint for the action given to the `define_spmd_block` function is to accept a `spmd_block` as first parameter.

**Public Functions**

```cpp
spmd_block ()
spmd_block (std::string const &name, std::size_t images_per_locality, std::size_t num_images, std::size_t image_id)
std::size_t get_images_per_locality () const
std::size_t get_num_images () const
std::size_t this_image () const
void sync_all () const

hpx::future<void> sync_all (hpx::launch::async_policy const &) const
void sync_images (std::set<std::size_t> const &images) const
void sync_images (std::vector<std::size_t> const &input_images) const
```

```cpp
template <typename Iterator>
std::enable_if<traits::is_input_iterator<Iterator>::value>::type sync_images (Iterator begin, Iterator end)

template <typename... I>
std::enable_if<util::detail::all_of<typename std::is_integral<I>::value>::type sync_images (I... i)

hpx::future<void> sync_images (hpx::launch::async_policy const &policy, std::set<std::size_t> const &images) const
hpx::future<void> sync_images (hpx::launch::async_policy const &policy, std::vector<std::size_t> const &input_images) const
```

```cpp
template <typename Iterator>
std::enable_if<traits::is_input_iterator<Iterator>::value, hpx::future<void>>::type sync_images (hpx::launch::async_policy, Iterator begin, Iterator end) const

template <typename... I>
```
std::enable_if<std::detail::all_of<typename std::is_integral<I>::type...>::value, hpx::future<void>>::type sync_images(hpx::launch::async_policy const& policy, I... i)

Private Types

using barrier_type = hpx::lcos::barrier
using table_type = std::map<std::set<std::size_t>, std::shared_ptr<barrier_type>>

Private Functions

template <typename Archive>
void serialize(Archive&, unsigned)

Private Members

std::string name_
std::size_t images_per_locality_
std::size_t num_images_
std::size_t image_id_
hpx::util::jenkins_hash hash_
std::shared_ptr<hpx::lcos::barrier> barrier_

private_type barriers_

Friends

friend hpx::lcos::hpx::serialization::access

coroutines

#include <compatibility/hpx/runtime/threads/thread Enums.hpp>
#include <compatibility/hpx/runtime/threads/thread Id Type.hpp>
#include <compatibility/hpx/runtime/threads/coroutines/coroutine.hpp>
#include <compatibility/hpx/runtime/threads/coroutines/coroutine_fwd.hpp>
#include <hpx/coroutines/thread Enums.hpp>

namespace hpx


namespace threads


 Enums

def thread_state_enum

The thread_state_enum enumerator encodes the current state of a thread instance.

Values:

unknown = 0
active = 1
    thread is currently active (running, has resources)
pending = 2
    thread is pending (ready to run, but no hardware resource available)
suspended = 3
    thread has been suspended (waiting for synchronization event, but still known and under control of the thread-manager)
depleted = 4
    thread has been depleted (deeply suspended, it is not known to the thread-manager)
terminated = 5
    thread has been stopped an may be garbage collected
staged = 6
    this is not a real thread state, but allows to reference staged task descriptions, which eventually will be converted into thread objects
pending_do_not_schedule = 7
pending_boost = 8

def thread_priority

This enumeration lists all possible thread-priorities for HPX threads.

Values:

thread_priority_unknown = -1
thread_priority_default = 0
    Will assign the priority of the task to the default (normal) priority.
thread_priority_low = 1
    Task goes onto a special low priority queue and will not be executed until all high/normal priority tasks are done, even if they are added after the low priority task.
thread_priority_normal = 2
    Task will be executed when it is taken from the normal priority queue, this is usually a first in-first-out ordering of tasks (depending on scheduler choice). This is the default priority.
thread_priority_high_recursive = 3
    The task is a high priority task and any child tasks spawned by this task will be made high priority as well - unless they are specifically flagged as non default priority.
thread_priority_boost = 4
    Same as thread_priority_high except that the thread will fall back to thread_priority_normal if resumed after being suspended.
thread_priority_high = 5
    Task goes onto a special high priority queue and will be executed before normal/low priority tasks are taken (some schedulers modify the behavior slightly and the documentation for those should be consulted).
thread_priority_bound = 6
Task goes onto a special high priority queue and will never be stolen by another thread after initial assignment. This should be used for thread placement tasks such as OpenMP type for loops.

enum thread_state_ex_enum
The thread_state_ex_enum enumerator encodes the reason why a thread is being restarted
Values:

wait_unknown = 0
wait_signaled = 1
The thread has been signaled.
wait_timeout = 2
The thread has been reactivated after a timeout
wait_terminate = 3
The thread needs to be terminated.
wait_abort = 4
The thread needs to be aborted.

enum thread_stacksize
A thread_stacksize references any of the possible stack-sizes for HPX threads.
Values:

thread_stacksize_unknown = -1
thread_stacksize_small = 1
use small stack size
thread_stacksize_medium = 2
use medium sized stack size
thread_stacksize_large = 3
use large stack size
thread_stacksize_huge = 4
use very large stack size
thread_stacksize_nostack = 5
this thread does not suspend (does not need a stack)
thread_stacksize_current = 6
use size of current thread’s stack

enum thread_schedule_hint_mode
The type of hint given when creating new tasks.
Values:

thread_schedule_hint_mode_none = 0
thread_schedule_hint_mode_thread = 1
thread_schedule_hint_mode_numa = 2
Functions

char const *get_thread_state_name (thread_state_enum state)
Returns the name of the given state.

Get the readable string representing the name of the given thread_state constant.

Parameters
• state: this represents the thread state.

cchar const *get_thread_priority_name (thread_priority priority)
Return the thread priority name.

Get the readable string representing the name of the given thread_priority constant.

Parameters
• this: represents the thread priority.

cchar const *get_thread_state_ex_name (thread_state_ex_enum state)
Get the readable string representing the name of the given thread_state_ex_enum constant.

cchar const *get_thread_state_name (thread_state state)
Get the readable string representing the name of the given thread_state constant.

cchar const *get_stack_size_name (std::ptrdiff_t size)
Returns the stack size name.

Get the readable string representing the given stack size constant.

Parameters
• size: this represents the stack size

struct thread_schedule_hint

Public Functions

thread_schedule_hint ()

thread_schedule_hint (std::int16_t thread_hint)

thread_schedule_hint (thread_schedule_hint_mode mode, std::int16_t hint)

Public Members

thread_schedule_hint_mode mode
std::int16_t hint

#include <hpx/coroutines/coroutine.hpp>

namespace hpx

namespace threads

namespace coroutines

class coroutine
Public Types

```cpp
using impl_type = detail::coroutine_impl
using thread_id_type = impl_type::thread_id_type
using result_type = impl_type::result_type
using arg_type = impl_type::arg_type
using functor_type = util::unique_function_nonser<result_type (arg_type)>
```

Public Functions

```cpp
coroutine (functor_type &&f, thread_id_type id, std::ptrdiff_t stack_size = detail::default_stack_size)
coroutine (coroutine const &src)
coroutine &operator= (coroutine const &src)
coroutine (coroutine &&src)
coroutine &operator= (coroutine &&src)
thread_id_type get_thread_id () const
std::size_t get_thread_data () const
std::size_t set_thread_data (std::size_t data)
void rebind (functor_type &&f, thread_id_type id)
result_type operator () (arg_type arg = arg_type())
bool is_ready () const
std::ptrdiff_t get_available_stack_space ()
impl_type *impl ()
```

Private Members

```cpp
impl_type impl_
```

#include <hpx/coroutines/thread_id_type.hpp>

namespace hpx

namespace threads

```

Variables

HPX_CONSTEXPR_OR_CONST thread_id hpx::threads::invalid_thread_id
```

struct thread_id
Public Functions

constexpr thread_id()
constexpr thread_id(thread_id_repr thrd)
thread_id(thread_id const&)
thread_id& operator=(thread_id const&)
constexpr operator bool() const
constexpr thread_id_repr get() const
HPX_CXX14_CONSTEXPR void hpx::threads::thread_id::reset()

Private Types

using thread_id_repr = void *

Private Members

thread_id_repr thrd_

Friends

constexpr bool operator== (std::nullptr_t, thread_id const & rhs)
constexpr bool operator!= (std::nullptr_t, thread_id const & rhs)
constexpr bool operator== (thread_id const & lhs, std::nullptr_t)
constexpr bool operator!= (thread_id const & lhs, std::nullptr_t)
constexpr bool operator== (thread_id const & lhs, thread_id const & rhs)
constexpr bool operator!= (thread_id const & lhs, thread_id const & rhs)
HPX_CXX14_CONSTEXPR bool operator<(thread_id const & lhs, thread_id const & rhs)
HPX_CXX14_CONSTEXPR bool operator>(thread_id const & lhs, thread_id const & rhs)
HPX_CXX14_CONSTEXPR bool operator<=(thread_id const & lhs, thread_id const & rhs)
HPX_CXX14_CONSTEXPR bool operator>=(thread_id const & lhs, thread_id const & rhs)

template <typename Char, typename Traits>
std::basic_ostream<Char, Traits>& operator<< (std::basic_ostream<Char, Traits>& os, thread_id const & id)

#include <hpx/coroutines/coroutine_fwd.hpp>
#include <hpx/coroutines/stackless_coroutine.hpp>

namespace hpx

namespace threads

Chapter 2. What’s so special about HPX?
namespace coroutines

class stackless_coroutine

Public Types

using thread_id_type = hpx::threads::thread_id
using result_type = std::pair<thread_state_enum, thread_id_type>
using arg_type = thread_state_ex_enum
using functor_type = util::unique_function_nonser<result_type (arg_type)>

Public Functions

stackless_coroutine (functor_type &&f, thread_id_type id, std::ptrdiff_t stack_size = default_stack_size)
~stackless_coroutine ()
stackless_coroutine (stackless_coroutine const &src)
stackless_coroutine &operator= (stackless_coroutine const &src)
stackless_coroutine (stackless_coroutine &&src)
stackless_coroutine &operator= (stackless_coroutine &&src)
thread_id_type get_thread_id () const
std::size_t get_thread_data () const
std::size_t set_thread_data (std::size_t data)
void rebind (functor_type &&f, thread_id_type id)
void reset_tss ()
void reset ()

stackless_coroutine::result_type operator() (arg_type arg = arg_type())

operator bool () const

bool is_ready () const
std::ptrdiff_t get_available_stack_space ()
std::size_t &get_continuation_recursion_count ()

Protected Attributes

functor_type f_
context_state state_
thread_id_type id_
std::size_t thread_data_
std::size_t continuation_recursion_count_
Private Types

enum context_state
    Values:
    ctx_running
    ctx_ready
    ctx_exited

Private Functions

bool running() const

bool exited() const

Private Members

HPX_STATIC_CONSTEXPR std::ptrdiff_t hpx::threads::coroutines::stackless_coroutine::default_stack_size

Friends

friend hpx::threads::coroutines::reset_on_exit

struct reset_on_exit

Public Functions

reset_on_exit(stackless_coroutine &this__)

~reset_on_exit()

Public Members

stackless_coroutine &this_

concurrency

#include <compatibility/hpx/util/spinlock_pool.hpp>
#include <compatibility/hpx/util/itt_notify.hpp>
#include <compatibility/hpx/util/spinlock.hpp>
#include <compatibility/hpx/util/lockfree/freelist.hpp>
#include <compatibility/hpx/util/lockfree/concurrentqueue.hpp>
#include <compatibility/hpx/util/lockfree/deque.hpp>
#include <compatibility/hpx/compat/barrier.hpp>
#include <hpx/concurrency/thread_name.hpp>
#include <hpx/concurrency/concurrentqueue.hpp>
Defines

MOODYCAMEL_THREADLOCAL
MOODYCAMEL.Exceptions_ENABLED
MOODYCAMEL_TRY
MOODYCAMEL_CATCH(...)
MOODYCAMEL_RETHROW
MOODYCAMEL_THROW(expr)
MOODYCAMEL_NOEXCEPT
MOODYCAMEL_NOEXCEPT_CTOR(type, valueType, expr)
MOODYCAMEL_NOEXCEPT_ASSIGN(type, valueType, expr)
MOODYCAMEL_DELETE_FUNCTION

namespace moodycamel

Functions

template <typename T, typename Traits>
void swap(Typename ConcurrentQueue<T, Traits>::ImplicitProducerKVP &a, Typename ConcurrentQueue<T, Traits>::ImplicitProducerKVP &b)

template <typename T, typename Traits>
void swap(ConcurrentQueue<T, Traits> &a, ConcurrentQueue<T, Traits> &b)
void swap(ProducerToken &a, ProducerToken &b)
void swap(ConsumerToken &a, ConsumerToken &b)

template <typename T, typename Traits = ConcurrentQueueDefaultTraits>
class ConcurrentQueue

Public Types

typedef moodycamel::ProducerToken producer_token_t
typedef moodycamel::ConsumerToken consumer_token_t
typedef Traits::index_t index_t
typedef Traits::size_t size_t

Public Functions

ConcurrentQueue(size_t capacity = 6 * BLOCK_SIZE)
ConcurrentQueue(size_t minCapacity, size_t maxExplicitProducers, size_t maxImplicitProducers)
~ConcurrentQueue()
ConcurrentQueue(ConcurrentQueue const&)
ConcurrentQueue &operator=(ConcurrentQueue const&)

2.9. API reference 425
HPX Documentation, 1.4.1

**ConcurrentQueue** (*ConcurrentQueue &other*)

*ConcurrentQueue &operator* = (*ConcurrentQueue &other*)

void **swap** (*ConcurrentQueue &other*)

bool **enqueue** (T const &item)

bool **enqueue** (T &item)

bool **enqueue** (*producer_token_t const &token, T const &item)

bool **enqueue** (*producer_token_t const &token, T &item)

**template** <tyename l>

bool **enqueue_bulk** (It itemFirst, size_t count)

**template** <tyename l>

bool **enqueue_bulk** (*producer_token_t const &token, It itemFirst, size_t count)

bool **try_enqueue** (T const &item)

bool **try_enqueue** (T &item)

bool **try_enqueue** (*producer_token_t const &token, T const &item)

bool **try_enqueue** (*producer_token_t const &token, T &item)

**template** <tyename l>

bool **try_enqueue_bulk** (It itemFirst, size_t count)

**template** <tyename l>

bool **try_enqueue_bulk** (*producer_token_t const &token, It itemFirst, size_t count)

**template** <tyename U>

bool **try_dequeue** (U &item)

**template** <tyename U>

bool **try_dequeue_non_interleaved** (U &item)

**template** <tyename U>

bool **try_dequeue** (*consumer_token_t &token, U &item)

**template** <tyename l>

size_t **try_dequeue_bulk** (It itemFirst, size_t max)

**template** <tyename l>

size_t **try_dequeue_bulk** (*consumer_token_t &token, It itemFirst, size_t max)

**template** <tyename U>

bool **try_dequeue_from_producer** (*producer_token_t const &producer, U &item)

**template** <tyename l>

size_t **try_dequeue_bulk_from_producer** (*producer_token_t const &producer, It itemFirst, size_t max)

**Public Static Functions**

static bool **is_lock_free** ()
Public Static Attributes

const size_t BLOCK_SIZE = static_cast<size_t>(Traits::BLOCK_SIZE)
const size_t EXPLICIT_BLOCK_EMPTY_COUNTER_THRESHOLD = static_cast<size_t>(Traits::EXPLICIT_BLOCK_EMPTY_COUNTER_THRESHOLD)
const size_t EXPLICIT_INITIAL_INDEX_SIZE = static_cast<size_t>(Traits::EXPLICIT_INITIAL_INDEX_SIZE)
const size_t IMPLICIT_INITIAL_INDEX_SIZE = static_cast<size_t>(Traits::IMPLICIT_INITIAL_INDEX_SIZE)
const size_t INITIAL_IMPLICIT_PRODUCER_HASH_SIZE = static_cast<size_t>(Traits::INITIAL_IMPLICIT_PRODUCER_HASH_SIZE)

const std::uint32_t EXPLICIT_CONSUMER_CONSUMPTION_QUOTA_BEFORE_ROTATE = static_cast<std::uint32_t>(Traits::EXPLICIT_CONSUMER_CONSUMPTION_QUOTA_BEFORE_ROTATE)

const size_t MAX_SUBQUEUE_SIZE = (details::const_numeric_max<size_t>::value - static_cast<size_t>(Traits::MAX_SUBQUEUE_SIZE)) ? details::const_numeric_max<size_t>::value : ((static_cast<size_t>(Traits::MAX_SUBQUEUE_SIZE) + (BLOCK_SIZE - 1)) / BLOCK_SIZE * BLOCK_SIZE)

Private Types

enum AllocationMode
Values:
  CanAlloc
  CannotAlloc

enum InnerQueueContext
Values:
  implicit_context = 0
  explicit_context = 1

Private Functions

ConcurrentQueue &swap_internal (ConcurrentQueue &other)

template <AllocationMode canAlloc, typename U>
bool inner_enqueue (producer_token_t const &token, U &&element)

template <AllocationMode canAlloc, typename U>
bool inner_enqueue (U &&element)

template <AllocationMode canAlloc, typename It>
bool inner_enqueue_bulk (producer_token_t const &token, It itemFirst, size_t count)

template <AllocationMode canAlloc, typename It>
bool inner_enqueue_bulk (It itemFirst, size_t count)

bool update_current_producer_after_rotation (consumer_token_t &token)

void populate_initial_block_list (size_t blockCount)

Block *try_get_block_from_initial_pool ()

void add_block_to_free_list (Block *block)

void add_blocks_to_free_list (Block *block)

Block *try_get_block_from_free_list ()

template <AllocationMode canAlloc>
Block *requisition_block ()

ProducerBase *recycle_or_create_producer (bool isExplicit)
ProducerBase *recycle_or_create_producer (bool isExplicit, bool &recycled)

ProducerBase *add_producer (ProducerBase *producer)

void reown_producers ()

void populate_initial_implicit_producer_hash ()

void swap_implicit_producer_hashes (ConcurrentQueue &other)

ImplicitProducer *get_or_add_implicit_producer ()

Private Members

std::atomic<ProducerBase *> producerListTail
std::atomic<std::uint32_t> producerCount
std::atomic<size_t> initialBlockPoolIndex
Block *initialBlockPool
size_t initialBlockPoolSize
FreeList<Block> freeList
std::atomic<ImplicitProducerHash *> implicitProducerHash
std::atomic<size_t> implicitProducerHashCount
ImplicitProducerHash initialImplicitProducerHash
std::array<ImplicitProducerKVP, INITIAL_IMPLICIT_PRODUCER_HASH_SIZE> initialImplicitProducerHashEntries
std::atomic_flag implicitProducerHashResizeInProgress
std::atomic<std::uint32_t> nextExplicitConsumerId
std::atomic<std::uint32_t> globalExplicitConsumerOffset

Private Static Functions

template <typename U>
static U *create_array (size_t count)

template <typename U>
static void destroy_array (U *p, size_t count)

template <typename U>
static U *create ()

template <typename U, typename A1>
static U *create (A1 &&a1)

template <typename U>
static void destroy (U *p)

Friends

friend moodycamel::ProducerToken
friend moodycamel::ConsumerToken
friend moodycamel::ExplicitProducer
friend moodycamel::ImplicitProducer
friend moodycamel::ConcurrentQueueTests

template <typename XT, typename XTraits>
void swap(typename ConcurrentQueue<XT, XTraits>::ImplicitProducerKVP&, typename ConcurrentQueue<XT, XTraits>::ImplicitProducerKVP&)

struct Block

Public Functions

template<> Block ()
template <InnerQueueContext context>
bool is_empty () const

template <InnerQueueContext context>
bool set_empty (index_t i)

template <InnerQueueContext context>
bool set_many_empty (index_t i, size_t count)

template <InnerQueueContext context>
void set_all_empty ()

template <>
void reset_empty ()

T *operator[] (index_t idx)

T const *operator[] (index_t idx) const

Public Members

template <>
char elements[sizeof(T) * BLOCK_SIZE]

template<>
details::max_align_t dummy

template<> Block *next

template<> std::atomic<size_t> elementsCompletelyDequeued

std::atomic<bool> moodycamel::ConcurrentQueue< T, Traits >::Block::emptyFlags[BLOCK_SIZE<=EXPLICIT_BLOCK_EMPTY_COUNTER_THRESHOLD ? BLOCK_SIZE :1]

template <>
std::atomic<std::uint32_t> freeListRefs

HPX Documentation, 1.4.1

std::atomic<Block *> freeListNext

template<> 
std::atomic<bool> shouldBeOnFreeList

template<> bool dynamicallyAllocated

Private Members

union moodycamel::ConcurrentQueue::Block::*8 moodycamel::ConcurrentQueue< T, Traits >::Block::*9

struct ExplicitProducer : public moodycamel::ConcurrentQueue<T, Traits>::ProducerBase

Public Functions

template<> ExplicitProducer (ConcurrentQueue *parent)

template<> ~ExplicitProducer ()

template <AllocationMode allocMode, typename U> template<> bool enqueue (U &&element)

template <typename U> template<> bool dequeue (U &element)

template <AllocationMode allocMode, typename It> template<> bool enqueue_bulk (It itemFirst, size_t count)

template <typename It> template<> size_t dequeue_bulk (It &itemFirst, size_t max)

Private Functions

template<> bool new_block_index (size_t numberOfFilledSlotsToExpose)

Private Members

std::atomic<BlockIndexHeader *> blockIndex

template<> size_t pr_blockIndexSlotsUsed

template<> size_t pr_blockIndexSize

template<> size_t pr_blockIndexFront

template<> BlockIndexEntry *pr_blockIndexEntries
template<> 
void *pr_blockIndexRaw

struct BlockIndexEntry

Public Members

template<>
index_t base

template<> 
Block *block

struct BlockIndexHeader

Public Members

template<>
size_t size

template<> 
std::atomic<size_t> front

template<> 
BlockIndexEntry *entries

template<> 
void *prev

template< typename N>
struct FreeList

Public Functions

template<> 
FreeList ()

template<> 
FreeList (FreeList &&other)

template<> 
void swap (FreeList &other)

template<> 
FreeList (FreeList const &)

template<> 
FreeList &operator= (FreeList const &)

template<> 
void add (N *node)

template<> 
N *try_get ()

template<> 
N *head_unsafe () const
Private Functions

template<> void add_knowing_refcount_is_zero (N *node)

Private Members

template<> std::atomic<N *> freeListHead

Private Static Attributes

template<> const std::uint32_t REFS_MASK = 0x7FFFFFFF

template<> const std::uint32_t SHOULD_BE_ON_FREELIST = 0x80000000

template <typename N>
struct FreeListNode

Public Functions

template<> FreeListNode ()

Public Members

template<> std::atomic<std::uint32_t> freeListRefs

template<> std::atomic<N *> freeListNext

struct ImplicitProducer : public moodycamel::ConcurrentQueue<T, Traits>::ProducerBase

Public Functions

template<> ImplicitProducer (ConcurrentQueue *parent)

template<> ~ImplicitProducer ()

template <AllocationMode allocMode, typename U>
template<> bool enqueue (U &&element)

template <typename U>
template<> bool dequeue (U &element)

template <AllocationMode allocMode, typename It>
template<> bool enqueue_bulk (It itemFirst, size_t count)

template <typename It>
template<> 
size_t dequeue_bulk(It &itemFirst, size_t max)

Private Functions

template <AllocationMode allocMode> 
template<> 
bool insert_block_index_entry(BlockIndexEntry *&idxEntry, index_t blockStartIndex)

template<> 
void rewind_block_index_tail()

template<> 
BlockIndexEntry *get_block_index_entry_for_index(index_t index) const

template<> 
size_t get_block_index_index_for_index(index_t index, BlockIndexHeader *&localBlockIndex) const

template<> 
bool new_block_index()

Private Members

template<> 
size_t nextBlockIndexCapacity

template<> 
std::atomic<BlockIndexHeader *> blockIndex

Private Static Attributes

template<> 
const index_t INVALID_BLOCK_BASE = 1

struct BlockIndexEntry

Public Members

template<> 
std::atomic<index_t> key

template<> 
std::atomic<Block *> value

struct BlockIndexHeader

Public Members

template<> 
size_t capacity

template<> 
std::atomic<size_t> tail

template<> 
BlockIndexEntry *entries
struct ImplicitProducerHash

Public Members

template<>
size_t capacity

template<>
ImplicitProducerKVP *entries

template<>
ImplicitProducerHash *prev

struct ImplicitProducerKVP

Public Functions

template<>
ImplicitProducerKVP ()

template<>
ImplicitProducerKVP (ImplicitProducerKVP &&other)

template<>
ImplicitProducerKVP &operator= (ImplicitProducerKVP &&other)

template<>
void swap (ImplicitProducerKVP &other)

Public Members

template<>
std::atomic<details::thread_id_t> key

template<>
ImplicitProducer *value

struct ProducerBase: public moodycamel::details::ConcurrentQueueProducerTypelessBase

Public Functions

template<>
ProducerBase (ConcurrentQueue *parent_, bool isExplicit_)

template<>
virtual ~ProducerBase ()

template <typename U>
dequeue (U &element)

template <typename It>
template<> 
size_t dequeue_bulk(It &itemFirst, size_t max)

template<> 
ProducerBase *next_prod() const

template<> 
szze_t size_approx() const

template<> 
index_t getTail() const

**Public Members**

template<> 
bool isExplicit

template<> 
ConcurrentQueue *parent

**Protected Attributes**

template<> 
std::atomic<index_t> tailIndex

template<> 
std::atomic<index_t> headIndex

template<> 
std::atomic<index_t> dequeueOptimisticCount

template<> 
std::atomic<index_t> dequeueOvercommit

template<> 
Block *tailBlock

**struct** ConcurrentQueueDefaultTraits

**Public Types**

typedef std::size_t size_t

typedef std::size_t index_t

**Public Static Functions**

static void *malloc(size_t size)

static void free(void *ptr)

**Public Static Attributes**

const size_t BLOCK_SIZE = 32

const size_t EXPLICIT_BLOCK_EMPTY_COUNTER_THRESHOLD = 32

const size_t EXPLICIT_INITIAL_INDEX_SIZE = 32

const size_t IMPLICIT_INITIAL_INDEX_SIZE = 32
const size_t INITIAL_IMPLICIT_PRODUCER_HASH_SIZE = 32
const std::uint32_t EXPLICIT_CONSUMER_CONSUMPTION_QUOTA_BEFORE_ROTATE = 256
const size_t MAX_SUBQUEUE_SIZE = details::const_numeric_max<size_t>::value

struct ConsumerToken

Public Functions

template <typename T, typename Traits>
ConsumerToken (ConcurrentQueue<T, Traits> &q)

template <typename T, typename Traits>
ConsumerToken (BlockingConcurrentQueue<T, Traits> &q)

ConsumerToken (ConsumerToken &&other)
ConsumerToken &operator= (ConsumerToken &&other)

void swap (ConsumerToken &other)

ConsumerToken (ConsumerToken const&)

ConsumerToken &operator= (ConsumerToken const&)

Private Members

std::uint32_t initialOffset
std::uint32_t lastKnownGlobalOffset
std::uint32_t itemsConsumedFromCurrent

details::ConcurrentQueueProducerTypelessBase *currentProducer

details::ConcurrentQueueProducerTypelessBase *desiredProducer

Friends

friend moodycamel::ConcurrentQueue
friend moodycamel::ConcurrentQueueTests

struct ProducerToken

Public Functions

template <typename T, typename Traits>
ProducerToken (ConcurrentQueue<T, Traits> &queue)

template <typename T, typename Traits>
ProducerToken (BlockingConcurrentQueue<T, Traits> &queue)

ProducerToken (ProducerToken &&other)

ProducerToken &operator= (ProducerToken &&other)

void swap (ProducerToken &other)

bool valid() const
~ProducerToken()

ProducerToken (ProducerToken const&)

ProducerToken &operator= (ProducerToken const&)

Protected Attributes

details::ConcurrentQueueProducerTypelessBase *producer

Friends

friend moodycamel::ConcurrentQueue
friend moodycamel::ConcurrentQueueTests

namespace details

Typedefs

typedef std::uintptr_t thread_id_t

typedef std::max_align_t std_max_align_t

Functions

static thread_id_t thread_id()
static bool() moodycamel::details::likely(bool x)
static bool() moodycamel::details::unlikely(bool x)
static size_t hash_thread_id(thread_id_t id)

template <typename T>
static bool circular_less_than (T a, T b)

template <typename U>
static char *align_for (char *ptr)

template <typename T>
static T ceil_to_pow_2 (T x)

template <typename T>
static void swap_relaxed (std::atomic<T> &left, std::atomic<T> &right)

template <typename T>
static T const &nomove (T const &x)

template <typename It>
static auto deref_noexcept (It &it)

Variables

const thread_id_t invalid_thread_id = 0
const thread_id_t invalid_thread_id2 = 1

template <bool use32>
struct _hash_32_or_64
Public Static Functions

```cpp
static std::uint32_t hash (std::uint32_t h)
```

```cpp
template <>
template<>
struct _hash_32_or_64<1>
```

Public Static Functions

```cpp
static std::uint64_t hash (std::uint64_t h)
```

```cpp
struct ConcurrentQueueProducerTypelessBase
```

Public Functions

```cpp
ConcurrentQueueProducerTypelessBase()
```

Public Members

```cpp
ConcurrentQueueProducerTypelessBase *next
std::atomic<bool> inactive
ProducerToken *token
```

```cpp
template <typename T>
struct const_numeric_max
```

Public Static Attributes

```cpp
? (static_cast<T>(1) << (sizeof(T) * CHAR_BIT - 1)) - static_cast<T>(1) : static_cast<T>(-1)
```

```cpp
union max_align_t
```

Public Members

```cpp
std_max_align_t x
long long y
void *z
```

```cpp
template <bool Enable>
struct nomove_if
```

Public Static Functions

```cpp
template <typename T>
static T const &eval (T const &x)
```

```cpp
template <>
template<>
struct nomove_if<false>
```
Public Static Functions

```cpp
template <typename U>
static auto eval(U &&x)
```

```cpp
template <>
template<>
struct static_is_lock_free<bool>
```

Public Types

```cpp
enum [anonymous]

Values:

value = ATOMIC_BOOL_LOCK_FREE
```

```cpp
template <typename U>
template<>
struct static_is_lock_free<U *>
```

Public Types

```cpp
enum [anonymous]

Values:

value = ATOMIC_POINTER_LOCK_FREE
```

```cpp
template <typename T>
struct static_is_lock_free_num
```

Public Types

```cpp
enum [anonymous]

Values:

value = 0
```

```cpp
template <>
template<>
struct static_is_lock_free_num<int>
```

Public Types

```cpp
enum [anonymous]

Values:

value = ATOMIC_INT_LOCK_FREE
```

```cpp
template <>
template<>
struct static_is_lock_free_num<long>
```

Public Types

```cpp
enum [anonymous]

Values:

value = ATOMIC_LONG_LOCK_FREE
```
template <>
template<>
struct static_is_lock_free_num<long long>

Public Types

enum [anonymous]

Values:

value = ATOMIC_LLONG_LOCK_FREE

template <>
template<>
struct static_is_lock_free_num<short>

Public Types

enum [anonymous]

Values:

value = ATOMIC_SHORT_LOCK_FREE

template <>
template<>
struct static_is_lock_free_num<signed char>

Public Types

enum [anonymous]

Values:

value = ATOMIC_CHAR_LOCK_FREE

template <typename thread_id_t>
struct thread_id_converter

Public Types

typedef thread_id_t thread_id_numeric_size_t

typedef thread_id_t thread_id_hash_t

Public Static Functions

static thread_id_hash_t prehash(thread_id_t const &x)

#include <hpx/concurrency/deque.hpp>

namespace boost

namespace lockfree
Enums

enum deque_status_type
Values:
stable
rpush
lpush

template <typename T, typename freelist_t = caching_freelist_t, typename Alloc = std::allocator<T>>
struct deque

Public Types

typedef deque_node<T> node

typedef node::pointer node_pointer

typedef node::atomic_pointer atomic_node_pointer

typedef node::tag_t tag_t

typedef deque_anchor<T> anchor

typedef anchor::pair anchor_pair

typedef anchor::atomic_pair atomic_anchor_pair

typedef Alloc::template rebind<node>::other node_allocator

typedef std::conditional<std::is_same<freelist_t, caching_freelist_t>::value, caching_freelist<node, node_allocator>, static_freelist<node, node_allocator>>::type pool

Public Functions

HPX_NON_COPYABLE (deque)

deque (std::size_t initial_nodes = 128)

~deque ()

bool empty () const

bool is_lock_free () const

bool push_left (T const &data)

bool push_right (T const &data)

bool pop_left (T &r)

bool pop_left (T *r)

bool pop_right (T &r)

bool pop_right (T *r)
Private Functions

node *alloc_node(node *lptr, node *rptr, T const &v, tag_t ltag = 0, tag_t rtag = 0)
void dealloc_node(node *n)
void stabilize_left(anchor_pair &lrs)
void stabilize_right(anchor_pair &lrs)
void stabilize(anchor_pair &lrs)

Private Members

anchor anchor_
pool pool_

BOOST_LOCKFREE_CACHELINE_BYTES - sizeof( anchor)]

template<>
char padding[padding_size]

template <typename T>
struct deque_anchor

Public Types

typedef deque_node<T> node
typedef node::pointer node_pointer
typedef node::atomic_pointer atomic_node_pointer
typedef node::tag_t tag_t
typedef tagged_ptr_pair<node, node> pair
typedef std::atomic<pair> atomic_pair

Public Functions

dequer anchor()
dequer anchor (deque_anchor const &p)
dequer anchor (pair const &p)
dequer anchor (node *lptr, node *rptr, tag_t status = stable, tag_t tag = 0)
pair lrs () volatile const
node *left () volatile const
node *right () volatile const
tag_t status () volatile const
tag_t tag () volatile const
bool cas (deque_anchor &expected, deque_anchor const &desired) volatile
bool cas (pair &expected, deque_anchor const &desired) volatile
bool cas (deque_anchor &expected, pair const &desired) volatile
bool cas (pair &expected, pair const &desired) volatile
bool operator==(volatile deque_anchor const &rhs) const
bool operator!=(volatile deque_anchor const &rhs) const
bool operator==(volatile pair const &rhs) const
bool operator!=(volatile pair const &rhs) const
bool is_lock_free() const

Private Members

atomic_pair pair_

template <typename T>
struct deque_node

Public Types

typedef detail::tagged_ptr<deque_node> pointer
typedef std::atomic<pointer> atomic_pointer
typedef pointer::tag_t tag_t

Public Functions

deqe_node ()
deqe_node (deque_node const &p)
deqe_node (deque_node *lp, deque_node *rp, T const &v, tag_t ltag = 0, tag_t rtag = 0)

Public Members

atomic_pointer left
atomic_pointer right
T data

#include <hpx/concurrency/barrier.hpp>

namespace hpx

namespace util

class barrier
Public Functions

```cpp
barrier (std::size_t number_of_threads)
~barrier ()
void wait ()
```

Private Types

typedef std::mutex mutex_type

Private Members

```cpp
static_cast<std::size_t>(1) << (CHAR_BIT * sizeof(std::size_t) - 1) 
std::size_t const number_of_threads_
std::size_t total_
mutex_type mtx_
std::condition_variable cond_
```

```cpp
#include <hpx/concurrency/cache_line_data.hpp>

template <typename Data>
```
```cpp
template<>
struct cache_aligned_data<Data, std::false_type>

```

Public Functions

```cpp
cache_aligned_data ()
cache_aligned_data (Data &&data)
cache_aligned_data (Data const &data)
```

Public Members

```cpp
Data data_
```

namespace hpx

```cpp
namespace threads
```

Functions

```cpp
constexpr std::size_t get_cache_line_size()
```

namespace util

Typedefs

```cpp
using hpx::util::cache_line_data = typedef cache_aligned_data<Data>
```

```cpp
template <typename Data, typename NeedsPadding = typename detail::needs_padding<Data>::type>
struct cache_aligned_data
```

Chapter 2. What’s so special about HPX?
Public Functions

`cache_aligned_data()`

`cache_aligned_data(Data &&data)`

`cache_aligned_data(Data const &data)`

Public Members

Data `data_`

```cpp
template<>
char cacheline_pad[get_cache_line_padding_size(sizeof(Data))]
```

```cpp
template <typename Data>
template<>
struct cache_aligned_data<Data, std::false_type>
```

Public Functions

`cache_aligned_data()`

`cache_aligned_data(Data &&data)`

`cache_aligned_data(Data const &data)`

Public Members

Data `data_`

```cpp
#include <hpx/concurrency/spinlock_pool.hpp>

namespace hpx

namespace util

```cpp
template <typename Tag, std::size_t N = HPX_HAVE_SPINLOCK_POOL_NUM>
class spinlock_pool
```

Public Static Functions

```cpp
static boost::detail::spinlock &spinlock_for (void const *pv)
```

Private Static Attributes

```cpp
cache_aligned_data<boost::detail::spinlock> pool_
```

```cpp
class scoped_lock
```
Public Functions

template<>
HPX_NON_COPYABLE (scoped_lock)

template<>
scoped_lock (void const *pv)

template<>
~scoped_lock()

template<>
void lock()

template<>
void unlock()

Private Members

template<>
boost::detail::spinlock &sp_

#include <hpx/concurrency/itt_notify.hpp>

Defines

HPX_ITT_SYNC_CREATE (obj, type, name)
HPX_ITT_SYNC_RENAME (obj, name)
HPX_ITT_SYNC_PREPARE (obj)
HPX_ITT_SYNC_CANCEL (obj)
HPX_ITT_SYNC_ACQUIRED (obj)
HPX_ITT_SYNC_RELEASED (obj)
HPX_ITT_SYNC_DESTROY (obj)
HPX_ITT_STACK_CREATE (ctx)
HPX_ITT_STACK_CALLEE.Enter (ctx)
HPX_ITT_STACK_CALLEE.Leave (ctx)
HPX_ITT_STACK_DESTROY (ctx)
HPX_ITT_FRAME_BEGIN (frame, id)
HPX_ITT_FRAME_END (frame, id)
HPX_ITT_MARK_CREATE (mark, name)
HPX_ITT_MARK_OFF (mark)
HPX_ITT_MARK (mark, parameter)
HPX_ITT_THREAD_SET_NAME (name)
HPX_ITT_THREAD_IGNORE ()
HPX_ITT_TASK_BEGIN (domain, name)
HPX_ITT_TASK_BEGIN_ID (domain, id, name)
HPX_ITT_TASK_END (domain)
HPX_ITT_DOMAIN_CREATE (name)
HPX_ITT_STRING_HANDLE_CREATE (name)
HPX_ITT_MAKE_ID (addr, extra)
HPX_ITT_ID_CREATE (domain, id)
HPX_ITT_ID_DESTROY (id)
HPX_ITT_HEAP_FUNCTION_CREATE (name, domain)
HPX_ITT_HEAP_ALLOCATE_BEGIN (f, size, initialized)
HPX_ITT_HEAP_ALLOCATE_END (f, addr, size, initialized)
HPX_ITT_HEAP_FREE_BEGIN (f, addr)
HPX_ITT_HEAP_FREE_END (f, addr)
HPX_ITT_HEAP_REALLOCATE_BEGIN (f, addr, new_size, initialized)
HPX_ITT_HEAP_REALLOCATE_END (f, addr, new_addr, new_size, initialized)
HPX_ITT_HEAP_INTERNAL_ACCESS_BEGIN ()
HPX_ITT_HEAP_INTERNAL_ACCESS_END ()
HPX_ITT_COUNTER_CREATE (name, domain)
HPX_ITT_COUNTER_CREATE_TYPED (name, domain, type)
HPX_ITT_COUNTER_SET_VALUE (id, value_ptr)
HPX_ITT_COUNTER_DESTROY (id)
HPX_ITT_METADATA_ADD (domain, id, key, data)

Typedefs

typedef void *__itt_heap_function

Functions

void itt_sync_create (void *, const char *, const char *)
void itt_sync_rename (void *, const char *)
void itt_sync_prepare (void *)
void itt_sync_acquired (void *)
void itt_sync_cancel (void *)
void itt_sync_releasing (void *)
void itt_sync_released (void *)
void itt_sync_destroy (void *)
void itt_stack_create ()
void itt_stack_enter (__itt_caller *)
void itt_stack_leave (__itt_caller *)
void itt_stack_destroy (___itt_caller *)
void itt_frame_begin (___itt_domain const *, ___itt_id *)
void itt_frame_end (___itt_domain const *, ___itt_id *)
int itt_mark_create (char const *)
void itt_mark_off (int)
void itt_mark (int, char const *)
void itt_thread_set_name (char const *)
void itt_thread_ignore ()
void itt_task_begin (___itt_domain const *, ___itt_id *, ___itt_string_handle *)
void itt_task_begin (___itt_domain const *, ___itt_string_handle *)
___itt_domain *itt_domain_create (char const *)
___itt_string_handle *itt_string_handle_create (char const *)
___itt_id *itt_make_id (void *, unsigned long)
void itt_id_create (___itt_domain const *, ___itt_id *)
void itt_id_destroy (___itt_id *)
__itt_heap_function itt_heap_function_create (const char *, const char *)
void itt_heap_allocate_begin (__itt_heap_function, std::size_t, int)
void itt_heap_allocate_end (__itt_heap_function, void **, std::size_t, int)
void itt_heap_free_begin (__itt_heap_function, void *)
void itt_heap_free_end (__itt_heap_function, void *)
void itt_heap_reallocate_begin (__itt_heap_function, void *, std::size_t, int)
void itt_heap_reallocate_end (__itt_heap_function, void *, void **, std::size_t, int)
void itt_heap_internal_access_begin ()
void itt_heap_internal_access_end ()
___itt_counter *itt_counter_create (char const *, char const *)
___itt_counter *itt_counter_create_typed (char const *, char const *, int)
void itt_counter_destroy (__itt_counter *)
void itt_counter_set_value (__itt_counter *, void *)
int itt_event_create (char const *, int)
int itt_event_start (int)
int itt_event_end (int)
void itt_metadata_add (___itt_domain *, ___itt_id *, ___itt_string_handle *, std::uint64_t const &)
void itt_metadata_add (___itt_domain *, ___itt_id *, ___itt_string_handle *, double const &)
void itt_metadata_add (___itt_domain *, ___itt_id *, ___itt_string_handle *, char const *)
void itt_metadata_add (___itt_domain *, ___itt_id *, ___itt_string_handle *, void const *)
namespace hpx

namespace util

namespace itt

Functions

void event_tick (event const&)

struct caller_context

Public Functions

caller_context (stack_context&)

~caller_context ()

struct counter

Public Functions

counter (char const *, char const *)

~counter ()

struct domain

Subclassed by hpx::util::itt::thread_domain

Public Functions

HPX_NON_COPYABLE (domain)

domain (char const *)

domain ()

struct event

Public Functions

event (char const *)

struct frame_context

Public Functions

frame_context (domain const&, id * = nullptr)

~frame_context ()

struct heap_allocate
Public Functions

```cpp
template <typename T>
heap_allocate (heap_function&, T**, std::size_t, int)
~heap_allocate()
```

```cpp
struct heap_free

Public Functions

heap_free (heap_function&, void*)
~heap_free()
```

```cpp
struct heap_function

Public Functions

heap_function (char const*, char const*)
~heap_function()
```

```cpp
struct heap_internal_access

Public Functions

heap_internal_access()
~heap_internal_access()
```

```cpp
struct id

Public Functions

id (domain const&, void*, unsigned long = 0)
~id()
```

```cpp
struct mark_context

Public Functions

mark_context (char const*)
~mark_context()
```

```cpp
struct mark_event

Public Functions

mark_event (event const&)
~mark_event()
```

```cpp
struct stack_context
```

Chapter 2. What’s so special about HPX?
Public Functions

stack_context()
~stack_context()

struct string_handle

Public Functions

string_handle(char const* = nullptr)

struct task

Public Functions

task(domain const&, string_handle const&, std::uint64_t)
task(domain const&, string_handle const&)
~task()

struct thread_domain: public hpx::util::itt::domain

Public Functions

HPX_NON_COPYABLE(thread_domain)

thread_domain()

struct undo_frame_context

Public Functions

undo_frame_context(frame_context const&)
~undo_frame_context()

struct undo_mark_context

Public Functions

undo_mark_context(mark_context const&)
~undo_mark_context()

#include <hpx/concurrency/spinlock.hpp>

namespace hpx

namespace util

struct spinlock

#include <spinlock.hpp> boost::mutex-compatible spinlock class
Public Types

typedef boost::detail::spinlock *native_handle_type

Public Functions

HPX_NON_COPYABLE (spinlock)
spinlock (char const * = nullptr)
~spinlock ()
void lock ()
bool try_lock ()
void unlock ()
native_handle_type native_handle ()

Private Members

boost::detail::spinlock m

serialization

#include <compatibility/hpx/runtime/serialization/shared_ptr.hpp>
#include <compatibility/hpx/runtime/serialization/multi_array.hpp>
#include <compatibility/hpx/runtime/serialization/string.hpp>
#include <compatibility/hpx/runtime/serialization/access.hpp>
#include <compatibility/hpx/runtime/serialization/list.hpp>
#include <compatibility/hpx/runtime/serialization/input_container.hpp>
#include <compatibility/hpx/runtime/serialization/base_object.hpp>
#include <compatibility/hpx/runtime/serialization/brace_initializable_fwd.hpp>
#include <compatibility/hpx/runtime/serialization/unordered_map.hpp>
#include <compatibility/hpx/runtime/serialization/output_container.hpp>
#include <compatibility/hpx/runtime/serialization/serialization_chunk.hpp>
#include <compatibility/hpx/runtime/serialization/binary_filter.hpp>
#include <compatibility/hpx/runtime/serialization/variant.hpp>
#include <compatibility/hpx/runtime/serialization/output_archive.hpp>
#include <compatibility/hpx/runtime/serialization/deque.hpp>
#include <compatibility/hpx/runtime/serialization/serialize_buffer.hpp>
#include <compatibility/hpx/runtime/serialization/valarray.hpp>
#include <compatibility/hpx/runtime/serialization/set.hpp>
#include <compatibility/hpx/runtime/serialization/input_archive.hpp>
#include <compatibility/hpx/runtime/serialization/dynamic_bitset.hpp>
#include <compatibility/hpx/runtime/serialization/optional.hpp>
#include <compatibility/hpx/runtime/serialization/intrusive_ptr.hpp>
#include <compatibility/hpx/runtime/serialization/vector.hpp>
#include <compatibility/hpx/runtime/serialization/tuple.hpp>
#include <compatibility/hpx/runtime/serialization/serialize.hpp>
#include <compatibility/hpx/runtime/serialization/bitset.hpp>
#include <compatibility/hpx/runtime/serialization/container.hpp>
#include <compatibility/hpx/runtime/serialization/serialization_fwd.hpp>
#include <compatibility/hpx/runtime/serialization/brace_initializable.hpp>
#include <compatibility/hpx/runtime/serialization/basic_archive.hpp>
#include <compatibility/hpx/runtime/serialization/map.hpp>
#include <compatibility/hpx/runtime/serialization/complex.hpp>
#include <compatibility/hpx/runtime/serialization/unique_ptr.hpp>
#include <compatibility/hpx/runtime/serialization/array.hpp>
#include <compatibility/hpx/traits/needs_automatic_registration.hpp>
#include <compatibility/hpx/traits/serialization_access_data.hpp>
#include <compatibility/hpx/traits/is_bitwise_serializable.hpp>
#include <compatibility/hpx/traits/brace_initializable_traits.hpp>
#include <compatibility/hpx/traits/polymorphic_traits.hpp>
#include <hpx/serialization/shared_ptr.hpp>

namespace hpx

namespace serialization

Functions

template <typename T>
void load (input_archive & ar, std::shared_ptr<T> & ptr, unsigned)

template <typename T>
void save(output_archive &ar, std::shared_ptr<T> const &ptr, unsigned)

#include <hpx/serialization/multi_array.hpp>
#include <hpx/serialization/string.hpp>

namespace hpx

namespace serialization

Functions

template <typename Char, typename CharTraits, typename Allocator>
void serialize(input_archive &ar, std::basic_string<Char, CharTraits, Allocator> &s, unsigned)

template <typename Char, typename CharTraits, typename Allocator>
void serialize(output_archive &ar, std::basic_string<Char, CharTraits, Allocator> const &s, unsigned)

#include <hpx/serialization/access.hpp>

template<typename T>
template<>
struct serialize_non_intrusive<T, typename std::enable_if<has_serialize_adl<T>::value>::type>

Public Static Functions

template <typename Archive>
static void call(Archive &ar, T &t, unsigned)

namespace hpx

namespace serialization

class access

Public Static Functions

template <class Archive, class T>
static void serialize(Archive &ar, T &t, unsigned)

template <typename Archive, typename T>
static void save_base_object(Archive &ar, T const &t, unsigned)

template <typename Archive, typename T>
static void load_base_object(Archive &ar, T &t, unsigned)

template <typename T>
static std::string get_name(T const *t)

template <class T>
class has_serialize
Public Static Attributes

```cpp
cconstexpr bool value = decltype(test<T>(0))::value
```

Private Static Functions

```cpp
template <class T1>
static std::false_type test (...)

template <class T1, class = decltype( std::declval<typename std::remove_const<T1>::type&>() .serialize(std::declval<
output_archive &>(), 0u))>
static std::true_type test (int)
```

```cpp
template <class T>
class serialize_dispatcher
```

Public Types

```cpp
template<>
using type = typename std::conditional::type

struct empty
```

Public Static Functions

```cpp
template <class Archive>
template<>
static void call (Archive& , T& , unsigned)
```

```cpp
struct intrusive_polymorphic
```

Public Static Functions

```cpp
template<>
static void call (hpx::serialization::input_archive &ar, T &t , unsigned)
```

```cpp
static void call (hpx::serialization::output_archive &ar, T const &t , unsigned)
```

```cpp
struct intrusive_usual
```

Public Static Functions

```cpp
template <class Archive>
template<>
static void call (Archive &ar, T &t , unsigned)
```

```cpp
struct non_intrusive
```

Public Static Functions

```cpp
template <class Archive>
template<>
static void call (Archive &ar, T &t , unsigned)
```

```cpp
template <typename T>
class has_serialize_adl
```
Public Static Attributes

```cpp
constexpr bool value = decltype(test<T*>(0))::value
```

Private Static Functions

```cpp
template <typename T1>
static std::false_type test(...)  
```

```cpp
template <typename T1, typename = decltype(serialize(std::declval<hpx::serialization::output_archive&>(), std::declval<typename std::remove_const<T1>::type&>(), 0u)))>
static std::true_type test(int)
```

```cpp
template <typename T>

struct serialize_non_intrusive<T, typename std::enable_if<has_serialize_adl<T>::value>::type>
```

Public Static Functions

```cpp
template <typename Archive>
static void call(Archive &ar, T &t, unsigned)
```

```cpp
#include <hpx/serialization/list.hpp>
namespace hpx

namespace serialization

Functions

```cpp
template <typename T, typename Allocator>
void serialize(input_archive &ar, std::list<T, Allocator> &ls, unsigned)
```

```cpp
template <typename T, typename Allocator>
void serialize(output_archive &ar, const std::list<T, Allocator> &ls, unsigned)
```

```cpp
#include <hpx/serialization/input_container.hpp>
namespace hpx

namespace serialization

```cpp
template <typename Container>
struct input_container: public hpx::serialization::erased_input_container

Public Functions

```cpp
input_container(Container const &cont, std::size_t inbound_data_size)
```

```cpp
input_container(Container const &cont, std::vector<serialization_chunk> const
*chunks, std::size_t inbound_data_size)
```

```cpp
void set_filter(binary_filter *filter)
```

```cpp
void load_binary(void *address, std::size_t count)
```
void load_binary_chunk (void *address, std::size_t count)

Public Members

Container const &cont_
std::size_t current_
std::unique_ptr<binary_filter> filter_
std::size_t decompressed_size_
std::vector<serialization_chunk> const *chunks_
std::size_t current_chunk_
std::size_t current_chunk_size_

Private Types

template<>
using access_traits = traits::serialization_access_data<Container>

Private Functions

std::size_t get_chunk_size (std::size_t chunk) const
std::uint8_t get_chunk_type (std::size_t chunk) const
chunk_data get_chunk_data (std::size_t chunk) const
std::size_t get_num_chunks () const

#include <hpx/serialization/base_object.hpp>

template <typename Derived, typename Base>
template<>
struct base_object_type<Derived, Base, std::true_type>

Public Functions

base_object_type (Derived &d)

template <class Archive>
void save (Archive &ar, unsigned) const

template <class Archive>
void load (Archive &ar, unsigned)

HPX.Serialization_SPLIT_MEMBER()

Public Members

Derived &d_

namespace hpx

namespace serialization
Functions

```cpp
template <typename Base, typename Derived>
base_object_type<Derived, Base> base_object (Derived &d)

template <typename D, typename B>
output_archive &operator<<(output_archive &ar, base_object_type<D, B> t)

template <typename D, typename B>
input_archive &operator>>(input_archive &ar, base_object_type<D, B> t)

template <typename D, typename B>
output_archive &operator& (output_archive &ar, base_object_type<D, B> t)

template <typename D, typename B>
input_archive &operator& (input_archive &ar, base_object_type<D, B> t)
```

```cpp
struct base_object_type

Public Functions

base_object_type (Derived &d)

template <typename Archive>
void serialize (Archive &ar, unsigned)

Public Members

Derived &d_
```

```cpp
template <typename Derived, typename Base>
template<>
struct base_object_type<Derived, Base, std::true_type>

Public Functions

base_object_type (Derived &d)

template <class Archive>
void save (Archive &ar, unsigned) const

template <class Archive>
void load (Archive &ar, unsigned)

HPX_SERIALIZATION_SPLIT_MEMBER()

Public Members

Derived &d_
```

```cpp
#include <hpx/serialization/brace_initializable_fwd.hpp>

#include <hpx/serialization/unordered_map.hpp>

namespace hpx

namespace serialization
```
Functions

```cpp
#include <hpx/serialization/output_container.hpp>
namespace hpx
namespace serialization

template <typename Key, typename Value, typename Hash, typename KeyEqual, typename Alloc>
void serialize (input_archive & ar, std::unordered_map<Key, Value, Hash, KeyEqual, Alloc> & t, unsigned)

template <typename Key, typename Value, typename Hash, typename KeyEqual, typename Alloc>
void serialize (output_archive & ar, const std::unordered_map<Key, Value, Hash, KeyEqual, Alloc> & t, unsigned)
```

```cpp
namespace hpx
namespace serialization

template <typename Container, typename Chunker>
struct filtered_output_container : public hpx::serialization::output_container<Container, Chunker>
{
    Public Types

    template<>
    using access_traits = traits::serialization_access_data<Container>
    template<>
    using base_type = output_container<Container, Chunker>

    Public Functions

    filtered_output_container (Container & cont, std::vector<serialization_chunk> * chunks = nullptr)

    ~filtered_output_container ()

    void flush ()

    void set_filter (binary_filter * filter)

    void save_binary (void const * address, std::size_t count)

    std::size_t save_binary_chunk (void const * address, std::size_t count)

    Protected Attributes

    std::size_t start_compressing_at_

    binary_filter * filter_

    template <typename Container, typename Chunker>
    struct output_container : public hpx::serialization::erased_output_container
    {
        Subclassed by hpx::serialization::filtered_output_container<Container, Chunker>
        Public Types

        template<>
        using access_traits = traits::serialization_access_data<Container>
    }
```
**Public Functions**

```cpp
output_container(Container &cont, std::vector<serialization_chunk> *chunks = nullptr)

~output_container()

void flush()

std::size_t get_num_chunks() const

void reset()

void set_filter(binary_filter *filter)

void save_binary(void const *address, std::size_t count)

std::size_t save_binary_chunk(void const *address, std::size_t count)

bool is_preprocessing() const
```

**Protected Attributes**

- Container &cont_
- std::size_t current_
- Chunker chunker_

```cpp
#include <hpx/serialization/serialization_chunk.hpp>

namespace hpx

namespace serialization

**Enums**

```cpp
enum chunk_type

Values:

chunk_type_index = 0

chunk_type_pointer = 1
```

**Functions**

```cpp
serialization_chunk create_index_chunk(std::size_t index, std::size_t size)

serialization_chunk create_pointer_chunk(void const *pos, std::size_t size, std::uint64_t rkey = 0)
```

**Public Members**

```cpp
std::size_t index_

void const *cpos_

void *pos_
```
struct serialization_chunk

Public Members

chunk_data data_
std::size_t size_
std::uint64_t rkey_
std::uint8_t type_

#include <hpx/serialization/binary_filter.hpp>

namespace hpx

namespace serialization

struct binary_filter

Public Functions

virtual void set_max_length (std::size_t size) = 0
virtual void save (void const *src, std::size_t src_count) = 0
virtual bool flush (void *dst, std::size_t dst_count, std::size_t &written) = 0
virtual std::size_t init_data (char const *buffer, std::size_t size, std::size_t buffer_size) = 0
virtual void load (void *dst, std::size_t dst_count) = 0
template <class T>
void serialize (T &, unsigned)

HPX_SERIALIZATION_POLYMORPHIC_ABSTRACT (binary_filter)

virtual ~binary_filter ()

#include <hpx/serialization/variant.hpp>

#include <hpx/serialization/output_archive.hpp>

namespace hpx

namespace serialization

struct output_archive : public hpx::serialization::basic_archive<output_archive>

Public Types

using base_type = basic_archive<output_archive>
**Public Functions**

```cpp
template <typename Container>
output_archive (Container &buffer, std::uint32_t flags = 0U,
                 std::vector<serialization_chunk> *chunks = nullptr,
                 binary_filter *filter = nullptr)

std::size_t bytes_written() const
std::size_t get_num_chunks() const
std::size_t current_pos() const

void reset()

void flush()

bool is_preprocessing() const
```

**Protected Functions**

```cpp
template <typename T>
void invoke_impl (T const &t)

template <typename T>
std::enable_if<std::is_integral<T>::value && !std::is_enum<T>::value>::type
save (T const &t)

template <typename T>
std::enable_if<std::is_integral<T>::value || std::is_enum<T>::value>::type
save (T t)

void save (float f)

void save (double d)

void save (char c)

void save (bool b)

template <typename T>
void save_bitwise (T const &t, std::false_type)

template <typename T>
void save_bitwise (T const &t, std::true_type)

template <typename T>
void save_nonintrusively_polymorphic (T const &t, std::false_type)

template <typename T>
void save_nonintrusively_polymorphic (T const &t, std::true_type)

template <typename T>
void save_integral (T val, std::false_type)

template <typename T>
void save_integral (T val, std::true_type)

template <class Promoted>
void save_integral_impl (Promoted l)

void save_binary (void const *address, std::size_t count)

void save_binary_chunk (void const *address, std::size_t count)
```
Protected Attributes

```cpp
std::unique_ptr<erased_output_container> buffer_
```

Private Static Functions

```cpp
static std::uint32_t make_flags (std::uint32_t flags, std::vector<serialization_chunk>* chunks)
```

Friends

```cpp
friend hpx::serialization::basic_archive< output_archive >
friend hpx::serialization::array
```

```cpp
#include <hpx/serialization/deque.hpp>
namespace hpx

namespace serialization

Functions

```cpp
template <typename T, typename Allocator>
void serialize (input_archive & ar, std::deque<T, Allocator> & d, unsigned)
```

```cpp
template <typename T, typename Allocator>
void serialize (output_archive & ar, std::deque<T, Allocator> const & d, unsigned)
```

```cpp
#include <hpx/serialization/serialize_buffer.hpp>
namespace hpx

namespace serialization

```

```cpp
template <typename T, typename Allocator = std::allocator<T>>
class serialize_buffer
```

Public Types

```cpp
enum init_mode
Values:
  copy = 0
  reference = 1
  take = 2
```

```cpp
using value_type = T
```
Public Functions

```cpp
serialize_buffer (allocator_type const &alloc = allocator_type())
serialize_buffer (std::size_t size, allocator_type const &alloc = allocator_type())
serialize_buffer (T *data, std::size_t size, init_mode mode = copy, allocator_type const &alloc = allocator_type())

template <typename Deallocator>
serialize_buffer (T *data, std::size_t size, allocator_type const &alloc, Deallocator const &dealloc)

template <typename Deleter>
serialize_buffer (T *data, std::size_t size, init_mode mode, Deleter const &deleter, allocator_type const &alloc = allocator_type())

template <typename Deleter>
serialize_buffer (T *data, std::size_t size, init_mode mode, Deleter const &deleter, allocator_type const &alloc = allocator_type())

T *data ()
T const *data () const
T *begin ()
T *end ()
T &operator[] (std::size_t idx)
T operator[] (std::size_t idx) const
boost::shared_array<T> data_array () const
std::size_t size () const
```

Private Types

```cpp
template<>
using allocator_type = Allocator
```

Private Functions

```cpp
template <typename Archive>
void save (Archive &ar, unsigned int const version) const
```
void load(Archive &ar, unsigned int const version)

**Private Members**

boost::shared_array<T> data_
std::size_t size_
Allocator alloc_

**Private Static Functions**

static void no_deleter (T *)

*template <typename Deallocator>*
static void deleter (T *p, Deallocator dealloc, std::size_t size)

**Friends**

friend hpx::serialization::hpx::serialization::access

bool operator==(serialize_buffer const &rhs, serialize_buffer const &lhs)

#include <hpx/serialization/valarray.hpp>
namespace hpx
namespace serialization

**Functions**

*template <typename T>*
void serialize(input_archive &ar, std::valarray<T> &arr, int)

*template <typename T>*
void serialize(output_archive &ar, std::valarray<T> const &arr, int)

#include <hpx/serialization/set.hpp>
namespace hpx
namespace serialization

**Functions**

*template <typename T, typename Compare, typename Allocator>*
void serialize(input_archive &ar, std::set<T, Compare, Allocator> &set, unsigned)

*template <typename T, typename Compare, typename Allocator>*
void serialize(output_archive &ar, std::set<T, Compare, Allocator> const &set, unsigned)
#include <hpx/serialization/input_archive.hpp>

namespace hpx

namespace serialization

    struct input_archive: public hpx::serialization::basic_archive<input_archive>

Public Types

using base_type = basic_archive<input_archive>

Public Functions

template <typename Container>
input_archive(Container &buffer, std::size_t inbound_data_size = 0, const std::vector<serialization_chunk> *chunks = nullptr)

    template <typename T>
void invoke_impl(T &t)

    template <typename T>
std::enable_if<!std::is_integral<T>::value && !std::is_enum<T>::value>::type load(T &t)

    template <typename T>
std::enable_if<std::is_integral<T>::value || std::is_enum<T>::value>::type load(T &t)

    void load(float &f)

    void load(double &d)

    void load(char &c)

    void load(bool &b)

    std::size_t bytes_read() const

    std::size_t current_pos() const

Private Functions

    template <typename T>
void load_bitwise(T &t, std::false_type)

    template <typename T>
void load_bitwise(T &t, std::true_type)

    template <class T>
void load_nonintrusively_polymorphic(T &t, std::false_type)

    template <class T>
void load_nonintrusively_polymorphic(T &t, std::true_type)

    template <typename T>
void load_integral(T &val, std::false_type)

    template <typename T>
void load_integral(T &val, std::true_type)
template <class Promoted>
void load_integral_impl (Promoted &l)

void load_binary (void *address, std::size_t count)

void load_binary_chunk (void *address, std::size_t count)

Private Members

std::unique_ptr<erased_input_container> buffer_

Friends

friend hpx::serialization::basic_archive< input_archive >
friend hpx::serialization::array

#include <hpx/serialization/dynamic_bitset.hpp>
#include <hpx/serialization/optional.hpp>
namespace hpx
namespace serialization

namespace serialization

Functions

template <typename T>
void save (output_archive &ar, hpx::util::optional<T> const &o, unsigned)

template <typename T>
void load (input_archive &ar, hpx::util::optional<T> &o, unsigned)

hpx::serialization::HPX_SERIALIZATION_SPLIT_FREE_TEMPLATE((template< typename T >), (hpx::util::optional< T >))

#include <hpx/serialization/intrusive_ptr.hpp>
#include <hpx/serialization/vector.hpp>
namespace hpx
namespace serialization

namespace serialization

Functions

template <typename Allocator>
void serialize (input_archive &ar, std::vector<bool, Allocator> &v, unsigned)

template <typename T, typename Allocator>
void serialize (input_archive &ar, std::vector<T, Allocator> &v, unsigned)

template <typename Allocator>
void serialize (output_archive &ar, std::vector<bool, Allocator> const &v, unsigned)

template <typename T, typename Allocator>
void serialize (output_archive &ar, std::vector<T, Allocator> const &v, unsigned)
#include <hpx/serialization/tuple.hpp>

namespace hpx

namespace serialization

Functions

template <typename Archive, typename... Ts>
void serialize(Archive & ar, hpx::util::tuple<Ts...> & t, unsigned int version)

template <typename Archive>
void serialize(Archive & ar, hpx::util::tuple<> & , unsigned)

#define include <hpx/serialization/serialize.hpp>

namespace hpx

namespace serialization

Functions

template <typename T>
output_archive & operator<<(output_archive & ar, T const & t)

template <typename T>
input_archive & operator>>(input_archive & ar, T & t)

#define include <hpx/serialization/bitset.hpp>

namespace hpx

namespace serialization

Functions

template <std::size_t N>
void serialize(input_archive & ar, std::bitset<N> & d, unsigned)

template <std::size_t N>
void serialize(output_archive & ar, std::bitset<N> const & hs, unsigned)
#include <hpx/serialization/container.hpp>

namespace hpx

namespace serialization

    struct erased_input_container
    
    Subclassed by hpx::serialization::input_container< Container >

    Public Functions
    
    virtual ~erased_input_container()
    virtual bool is_preprocessing() const
    virtual void set_filter(binary_filter *filter) = 0
    virtual void load_binary(void *address, std::size_t count) = 0
    virtual void load_binary_chunk(void *address, std::size_t count) = 0

    struct erased_output_container
    
    Subclassed by hpx::serialization::output_container< Container, Chunker >

    Public Functions
    
    virtual ~erased_output_container()
    virtual bool is_preprocessing() const
    virtual void set_filter(binary_filter *filter) = 0
    virtual void save_binary(void const *address, std::size_t count) = 0
    virtual std::size_t save_binary_chunk(void const *address, std::size_t count) = 0
    virtual void reset() = 0
    virtual std::size_t get_num_chunks() const = 0
    virtual void flush() = 0

#include <hpx/serialization/serialization_fwd.hpp>

Defines

HPX_SERIALIZATION_SPLIT_MEMBER()
HPX_SERIALIZATION_SPLIT_FREE(T)
HPX_SERIALIZATION_SPLIT_FREE_TEMPLATE(TEMPLATE, ARGS)

namespace hpx

    namespace serialization

2.9. API reference 469
Functions

```cpp
template <typename T>
output_archive &operator& (output_archive &ar, T const &t)

template <typename T>
input_archive &operator& (input_archive &ar, T &t)
```

#include <hpx/serialization/brace_initializable.hpp>

#include <hpx/serialization/basic_archive.hpp>

namespace hpx

namespace serialization

Enums

```cpp
eenum archive_flags
    Values:
    no_archive_flags = 0x00000000
    enable_compression = 0x00002000
    endian_big = 0x00004000
    endian_little = 0x00008000
    disable_array_optimization = 0x00010000
    disable_data_chunking = 0x00020000
    all_archive_flags = 0x0003e000
```

Functions

```cpp
void reverse_bytes (std::size_t size, char *address)

template <typename Archive>
void save_binary (Archive &ar, void const *address, std::size_t count)

template <typename Archive>
void load_binary (Archive &ar, void *address, std::size_t count)

template <typename Archive>
std::size_t current_pos (const Archive &ar)

template <typename Archive>
struct basic_archive
```

Public Functions

```cpp
virtual ~basic_archive ()

template <typename T>
void invoke (T &t)

bool enable_compression () const
```
bool endian_big() const
bool endian_little() const
bool disable_array_optimization() const
bool disable_data_chunking() const
std::uint32_t flags() const
bool is_preprocessing() const
std::size_t current_pos() const
void save_binary(void const *address, std::size_t count)
void load_binary(void *address, std::size_t count)
void reset()

template<typename T>
T &get_extra_data()

Public Static Attributes
const std::uint64_t npos = std::uint64_t(-1)

Protected Functions

basic_archive(std::uint32_t flags)
basic_archive(basic_archive const&)

basic_archive &operator=(basic_archive const&)

Protected Attributes

std::uint32_t flags_
std::size_t size_
detail::extra_archive_data extra_data_

#include <hpx/serialization/map.hpp>

namespace hpx

namespace serialization

Functions

template<typename Key, typename Value>
void serialize(input_archive &ar, std::pair<Key, Value> &t, unsigned)

template<typename Key, typename Value>
void serialize(output_archive &ar, const std::pair<Key, Value> &t, unsigned)

template <typename Key, typename Value, typename Comp, typename Alloc>
void serialize(input_archive &ar, std::map<Key, Value, Comp, Alloc> &t, unsigned)

template <typename Key, typename Value, typename Comp, typename Alloc>
void serialize(output_archive &ar, std::map<Key, Value, Comp, Alloc> const &t, unsigned)

#include <hpx/serialization/complex.hpp>

namespace hpx

namespace serialization

Functions

template <typename T>
void serialize(input_archive &ar, std::complex<T> &c, unsigned)

template <typename T>
void serialize(output_archive &ar, std::complex<T> const &c, unsigned)

#include <hpx/serialization/unique_ptr.hpp>

namespace hpx

namespace serialization

Functions

template <typename T>
void load(input_archive &ar, std::unique_ptr<T> &ptr, unsigned)

template <typename T>
void save(output_archive &ar, const std::unique_ptr<T> &ptr, unsigned)

#include <hpx/serialization/array.hpp>

namespace hpx

namespace serialization

Functions

template <class T>
array<T> make_array(T *begin, std::size_t size)

template <typename Archive, typename T, std::size_t N>
void serialize(Archive &ar, std::array<T, N> &a, const unsigned int)

template <typename T>
output_archive &operator<<(output_archive &ar, array<T> t)

template <typename T>
input_archive &operator>>(input_archive &ar, array<T> t)


```cpp
template <typename T>
output_archive &operator& (output_archive &ar, array<T> t)

template <typename T>
input_archive &operator& (input_archive &ar, array<T> t)

template <typename T, std::size_t N>
output_archive &operator<< (output_archive &ar, T (&t)[N])

template <typename T, std::size_t N>
input_archive &operator>> (input_archive &ar, T (&t)[N])

template <typename T, std::size_t N>
output_archive &operator& (output_archive &ar, T (&t)[N])

template <typename T, std::size_t N>
input_archive &operator& (input_archive &ar, T (&t)[N])

template <class T>
class array

Public Types

template<>
using value_type = T

Public Functions

array (value_type *, std::size_t s)
value_type *address () const
std::size_t count () const

template <class Archive>
void serialize_optimized (Archive &ar, unsigned int, std::false_type)

void serialize_optimized (output_archive &ar, unsigned int, std::true_type)

void serialize_optimized (input_archive &ar, unsigned int, std::true_type)

template <class Archive>
void serialize (Archive &ar, unsigned int v)

Private Members

value_type *m_t
std::size_t m_element_count
```

#include <hpx/serialization/traits/needs_automatic_registration.hpp>

#include <hpx/serialization/traits/serialization_access_data.hpp>

namespace hpx

namespace traits

2.9. API reference 473
template<typename Container>
struct default_serialization_access_data
    Subclassed by hpx::traits::serialization_access_data<Container>

Public Types

    template<>
    using preprocessing_only = std::false_type

Public Static Functions

    static bool is_preprocessing()
    static HPX_CXX14_CONSTEXPR void hpx::traits::default_serialization_access_data::write(Container & cont, std::size_t count, std::size_t current, void const * address)
    static HPX_CXX14_CONSTEXPR void hpx::traits::default_serialization_access_data::read(Container const & cont, std::size_t count, std::size_t current, void * address)
    static HPX_CXX14_CONSTEXPR void hpx::traits::default_serialization_access_data::reset(Container & cont)

    template<typename Container>
struct serialization_access_data : public hpx::traits::default_serialization_access_data<Container>
    Subclassed by hpx::traits::serialization_access_data<Container const>

Public Static Functions

    static std::size_t size(Container const & cont)
    static void resize(Container & cont, std::size_t count)
    static void write(Container & cont, std::size_t count, std::size_t current, void const * address)
    static bool flush(serialization::binary_filter * filter, Container &cont, std::size_t current, std::size_t &written)
    static HPX_CXX14_CONSTEXPR void hpx::traits::default_serialization_access_data::flush(serialization::binary_filter * filter, Container &cont, std::size_t current, std::size_t size, std::size_t &written)
    static std::size_t init_data(Container const & cont, serialization::binary_filter * filter, std::size_t current, std::size_t decompressed_size)
    static HPX_CXX14_CONSTEXPR void hpx::traits::default_serialization_access_data::init_data(Container const & cont, serialization::binary_filter * filter, std::size_t current, std::size_t decompressed_size)

#include <hpx/serialization/traits/is_bitwise_serializable.hpp>

Defines

HPX_IS_BITWISE_SERIALIZABLE(T)

#include <hpx/serialization/traits/brace_initializable_traits.hpp>

#include <hpx/serialization/traits/polymorphic_traits.hpp>

Defines

HPX_TRAITS_NONINTRUSIVE_POLYMORPHIC(Class)
HPX_TRAITS_NONINTRUSIVE_POLYMORPHIC_TEMPLATE (TEMPLATE, ARG_LIST)

HPX_TRAITS_SERIALIZED_WITH_ID (Class)

HPX_TRAITS_SERIALIZED_WITH_ID_TEMPLATE (TEMPLATE, ARG_LIST)

execution

#include <hpx/traits/is_execution_policy.hpp>

namespace hpx

namespace parallel

namespace execution

    template <typename T>
    struct is_async_execution_policy : public execution::detail::is_async_execution_policy<hpx::util::decay<T>::type>
    #include <is_execution_policy.hpp>
    Extension: Detect whether given execution policy makes algorithms asynchronous
    1. The type is_async_execution_policy can be used to detect asynchronous execution policies for the purpose of excluding function signatures from otherwise ambiguous overload resolution participation.
    2. If T is the type of a standard or implementation-defined execution policy, is_async_execution_policy<T> shall be publicly derived from integral_constant<bool, true>, otherwise from integral_constant<bool, false>.
    3. The behavior of a program that adds specializations for is_async_execution_policy is undefined.

    template <typename T>
    struct is_execution_policy : public execution::detail::is_execution_policy<hpx::util::decay<T>::type>
    #include <is_execution_policy.hpp>
    1. The type is_execution_policy can be used to detect execution policies for the purpose of excluding function signatures from otherwise ambiguous overload resolution participation.
    2. If T is the type of a standard or implementation-defined execution policy, is_execution_policy<T> shall be publicly derived from integral_constant<bool, true>, otherwise from integral_constant<bool, false>.
    3. The behavior of a program that adds specializations for is_execution_policy is undefined.

    template <typename T>
    struct is_parallel_execution_policy : public execution::detail::is_parallel_execution_policy<hpx::util::decay<T>::type>
    #include <is_execution_policy.hpp>
    Extension: Detect whether given execution policy enables parallelization
    1. The type is_parallel_execution_policy can be used to detect parallel execution policies for the purpose of excluding function signatures from otherwise ambiguous overload resolution participation.
    2. If T is the type of a standard or implementation-defined execution policy, is_parallel_execution_policy<T> shall be publicly derived from integral_constant<bool, true>, otherwise from integral_constant<bool, false>.
    3. The behavior of a program that adds specializations for is_parallel_execution_policy is undefined.

    template <typename T>
struct is_sequenced_execution_policy : public execution::detail::is_sequenced_execution_policy<hp::util::decay<T>::type>
#include <is_execution_policy.hpp> Extension: Detect whether given execution policy does not enable parallelization
1. The type is_sequenced_execution_policy can be used to detect non-parallel execution policies for the purpose of excluding function signatures from otherwise ambiguous overload resolution participation.
2. If \( T \) is the type of a standard or implementation-defined execution policy, is_sequenced_execution_policy<\( T \)> shall be publicly derived from integral_constant<bool, true>, otherwise from integral_constant<bool, false>.
3. The behavior of a program that adds specializations for is_sequenced_execution_policy is undefined.

#include <hpx/traits/is_executor_parameters.hpp>

template <typename Executor>
template<>
struct extract_executor_parameters<Executor, typename hpx::util::always_void<typename Executor::executor_parameters_type>::type>

Public Types

template<>
using type = typename Executor::executor_parameters_type

template< typename Parameters>
template<>
struct extract_has_variable_chunk_size<Parameters, typename hpx::util::always_void<typename Parameters::has_variable_chunk_size>::type>

Public Types

template<>
using type = typename Parameters::has_variable_chunk_size

namespace hpx

namespace parallel

namespace execution

Typedefs

using hpx::parallel::execution::is_executor_parameters_t = typedef typename is_executor_parameters<T>::type

template <typename Executor, typename Enable = void>
struct extract_executor_parameters

Public Types

template<>
using type = sequential_executor_parameters

template< typename Executor>
template<>
struct extract_executor_parameters<Executor, typename hpx::util::always_void<typename Executor::executor_parameters_type>::type>

Public Types

template<>
using type = typename Executor::executor_parameters_type

template <typename Parameters, typename Enable = void>
struct extract_has_variable_chunk_size

Public Types

template<>
using type = std::false_type

template <typename Parameters>
template<>
struct extract_has_variable_chunk_size<Parameters, typename hpx::util::always_void<Parameters>::type>

Public Types

template<>
using type = typename Parameters::has_variable_chunk_size

namespace traits

Typedefs

using hpx::traits::is_executor_parameters_t = typedef typename is_executor_parameters<Executor>::type

#include <hpx/traits/executor_traits.hpp>

namespace hpx

namespace parallel

namespace execution

template <typename Executor>
struct executor_context

Public Types

template<>
using type = typename std::decay::type

template <typename Executor>
struct executor_execution_category

Public Types

template<>
using type = hpx::util::detected_or_t<unsequenced_execution_tag, execution_category, Executor>
Private Types

template<>
using execution_category = typename T::execution_category

Public Types

template<>
using type = hpx::util::detected_or_t<
    typename executor_shape<Executor>::type,
    index_type,
    Executor>

Private Types

template<>
using index_type = typename T::index_type

Public Types

template<>
using type = hpx::util::detected_or_t<
    parallel::execution::static_chunk_size,
    parameters_type,
    Executor>

Private Types

template<>
using parameters_type = typename T::parameters_type

Public Types

template<>
using type = hpx::util::detected_or_t<
    std::size_t,
    shape_type,
    Executor>

Private Types

template<>
using shape_type = typename T::shape_type

namespace traits

Typedefs

using hpx::traits::executor_context_t = typedef typename executor_context<Executor>::type
using hpx::traits::executor_execution_category_t = typedef typename executor_execution_category<Executor>::type
using hpx::traits::executor_shape_t = typedef typename executor_shape<Executor>::type
using hpx::traits::executor_index_t = typedef typename executor_index<Executor>::type
using hpx::traits::executor_parameters_type_t = typedef typename executor_parameters_type<Executor>::type
#include <hpx/traits/is_timed_executor.hpp>

namespace hpx

namespace parallel

namespace execution

Typedefs

using hpx::parallel::execution::is_timed_executor_t = typedef typename is_timed_executor<T>::type

#include <hpx/traits/is_executor.hpp>

namespace hpx

namespace parallel

namespace execution

Typedefs

using hpx::parallel::execution::is_one_way_executor_t = typedef typename is_one_way_executor<T>::type
using hpx::parallel::execution::is_never_blocking_one_way_executor_t = typedef typename is_never_blocking_one_way_executor<T>::type
using hpx::parallel::execution::is_bulk_one_way_executor_t = typedef typename is_bulk_one_way_executor<T>::type
using hpx::parallel::execution::is_two_way_executor_t = typedef typename is_two_way_executor<T>::type
using hpx::parallel::execution::is_bulk_two_way_executor_t = typedef typename is_bulk_two_way_executor<T>::type

namespace traits

Typedefs

using hpx::traits::is_one_way_executor_t = typedef typename is_one_way_executor<T>::type
using hpx::traits::is_never_blocking_one_way_executor_t = typedef typename is_never_blocking_one_way_executor<T>::type
using hpx::traits::is_bulk_one_way_executor_t = typedef typename is_bulk_one_way_executor<T>::type
using hpx::traits::is_two_way_executor_t = typedef typename is_two_way_executor<T>::type
using hpx::traits::is_bulk_two_way_executor_t = typedef typename is_bulk_two_way_executor<T>::type
using hpx::traits::is_executor_any_t = typedef typename is_executor_any<T>::type

#include <hpx/parallel/execution_policy_fwd.hpp>

namespace hpx

namespace parallel
namespace execution

```cpp
template <typename Executor, typename Parameters>
struct parallel_policy_shim: public hpx::parallel::execution::parallel_policy

#include <execution_policy.hpp>
The class parallel_policy_shim is an execution policy type used as a unique type to disambiguate parallel algorithm overloading and indicate that a parallel algorithm’s execution may be parallelized.

```

```cpp
template <typename Executor, typename Parameters>
struct parallel_task_policy_shim: public hpx::parallel::execution::parallel_task_policy

#include <execution_policy.hpp>
Extension: The class parallel_task_policy_shim is an execution policy type used as a unique type to disambiguate parallel algorithm overloading based on combining a underlying parallel_task_policy and an executor and indicate that a parallel algorithm’s execution may be parallelized.

```

```cpp
template <typename Executor, typename Parameters>
struct sequenced_policy_shim: public hpx::parallel::execution::sequenced_policy

#include <execution_policy.hpp>
The class sequenced_policy is an execution policy type used as a unique type to disambiguate parallel algorithm overloading and require that a parallel algorithm’s execution may not be parallelized.

```

```cpp
template <typename Executor, typename Parameters>
struct sequenced_task_policy_shim: public hpx::parallel::execution::sequenced_task_policy

#include <execution_policy.hpp>
Extension: The class sequenced_task_policy_shim is an execution policy type used as a unique type to disambiguate parallel algorithm overloading based on combining a underlying sequenced_task_policy and an executor and indicate that a parallel algorithm’s execution may not be parallelized (has to run sequentially).

The algorithm returns a future representing the result of the corresponding algorithm when invoked with the sequenced_policy.

```cpp
#include <hpx/parallel/executors.hpp>
#include <hpx/parallel/execution.hpp>
#include <hpx/parallel/execution_policy.hpp>
namespace hpx

namespace parallel

namespace execution

Variables
```

```cpp
task_policy_tag HPX_CONSTEXPR_OR_CONST hpx::parallel::execution::task
Default sequential execution policy object.

```

```cpp
HPX_STATIC_CONSTEXPR sequenced_policy hpx::parallel::execution::seq
Default sequential execution policy object.

```

```cpp
HPX_STATIC_CONSTEXPR parallel_policy hpx::parallel::execution::par
Default parallel execution policy object.
```
Default vector execution policy object.

The class `parallel_policy` is an execution policy type used as a unique type to disambiguate parallel algorithm overloading and indicate that a parallel algorithm’s execution may be parallelized.

Subclassed by `hpx::parallel::execution::parallel_policy_shim<Executor, Parameters>`

### Public Types

**typedef** parallel_executor *executor_type*

The type of the executor associated with this execution policy.

**typedef execution::extract_executor_parameters<executor_type>::type *executor_parameters_type**

The type of the associated executor parameters object which is associated with this execution policy.

**typedef parallel_execution_tag *execution_category**

The category of the execution agents created by this execution policy.

### Public Functions

**parallel_task_policy operator() (task_policy_tag) const**

Create a new `parallel_policy` referencing a chunk size.

**Return** The new `parallel_policy`

**Parameters**

- `tag`: [in] Specify that the corresponding asynchronous execution policy should be used

**template <typename Executor>
rebind_executor<parallel_policy, Executor, executor_parameters_type>::type on (Executor &&exec) const**

Create a new `parallel_policy` referencing an executor and a chunk size.

**Return** The new `parallel_policy`

**Parameters**

- `exec`: [in] The executor to use for the execution of the parallel algorithm the returned execution policy is used with

**template <typename... Parameters, typename ParametersType = typename executor_parameters_join<Parameters...>::type
rebind_executor<parallel_policy, executor_type, ParametersType>::type with (Parameters&&... params) const**

Create a new `parallel_policy` from the given execution parameters

**Note** Requires: is_executor_parameters<Parameters>::value is true

**Return** The new `parallel_policy`

**Template Parameters**

- `Parameters`: The type of the executor parameters to associate with this execution policy.

**Parameters**

- `params`: [in] The executor parameters to use for the execution of the parallel algorithm the returned execution policy is used with.

**executor_type &executor()**

Return the associated executor object.
executor_type const &executor() const
    Return the associated executor object.

executor_parameters_type &parameters()
    Return the associated executor parameters object.

executor_parameters_type const &parameters() const
    Return the associated executor parameters object.

Private Functions

template<typename Archive>
void serialize(Archive &ar, const unsigned int version)

Private Members

executor_type exec_

executor_parameters_type params_

Friends

friend hpx::parallel::execution::hpx::serialization::access

template<typename Executor_, typename Parameters_>
struct rebind
    #include <execution_policy.hpp> Rebind the type of executor used by this execution policy.
    The execution category of Executor shall not be weaker than that of this execution policy

Public Types

typedef parallel_policy_shim<Executor_, Parameters_> type
    The type of the rebound execution policy.

struct parallel_task_policy
    #include <execution_policy.hpp> Extension: The class parallel_task_policy is an execution pol-
    icy type used as a unique type to disambiguate parallel algorithm overloading and indicate that a
    parallel algorithm’s execution may be parallelized.
    The algorithm returns a future representing the result of the corresponding algorithm when in-
    voked with the parallel_policy.
    Subclassed by hpx::parallel::execution::parallel_task_policy_shim< Executor, Parameters >

Public Types

typedef parallel_executor executor_type
    The type of the executor associated with this execution policy.

typedef execution::extract_executor_parameters<executor_type>::type executor_parameters_type
    The type of the associated executor parameters object which is associated with this execution
    policy

typedef parallel_execution_tag execution_category
    The category of the execution agents created by this execution policy.
Public Functions

`parallel_task_policy operator() (task_policy_tag) const`
Create a new `parallel_task_policy` from itself

**Return** The new `parallel_task_policy`

**Parameters**
- `tag`: [in] Specify that the corresponding asynchronous execution policy should be used

```cpp
template <typename Executor>
rebind_executor<parallel_task_policy, Executor, executor_parameters_type>::type on (Executor &&exec) const
```
Create a new `parallel_task_policy` from given executor

**Note** Requires: `is_executor<Executor>::value` is true

**Return** The new `parallel_task_policy`

**Template Parameters**
- `Executor`: The type of the executor to associate with this execution policy.

**Parameters**
- `exec`: [in] The executor to use for the execution of the parallel algorithm the returned execution policy is used with.

```cpp
template <typename... Parameters, typename ParametersType = typename executor_parameters_join<Parameters...>::type>
rebind_executor<parallel_task_policy, executor_type, ParametersType>::type with (Parameters&&... params) const
```
Create a new `parallel_policy_shim` from the given execution parameters

**Note** Requires: all parameters are executor_parameters, different parameter types can’t be duplicated

**Return** The new `parallel_policy_shim`

**Template Parameters**
- `Parameters`: The type of the executor parameters to associate with this execution policy.

**Parameters**
- `params`: [in] The executor parameters to use for the execution of the parallel algorithm the returned execution policy is used with.

```cpp
executor_type &executor ()
```
Return the associated executor object.

```cpp
executor_type const &executor () const
```
Return the associated executor object.

```cpp
executor_parameters_type &parameters ()
```
Return the associated executor parameters object.

```cpp
executor_parameters_type const &parameters () const
```
Return the associated executor parameters object.

Private Functions

```cpp
template <typename Archive>
void serialize (Archive &ar, const unsigned int version)
```
Private Members

executor_type exec_
executor_parameters_type params_

Friends

friend hpx::parallel::execution::hpx::serialization::access
template <typename Executor_, typename Parameters_>
struct rebind
#include <execution_policy.hpp> Rebind the type of executor used by this execution policy.
The execution category of Executor shall not be weaker than that of this execution policy

Public Types

typedef parallel_task_policy_shim<Executor_, Parameters_> type
The type of the rebound execution policy.

struct parallel_unsequenced_policy
#include <execution_policy.hpp> The class parallel_unsequenced_policy is an execution policy
type used as a unique type to disambiguate parallel algorithm overloading and indicate that a
parallel algorithm’s execution may be vectorized.

Public Types

typedef parallel_executor executor_type
The type of the executor associated with this execution policy.

typedef execution::extract_executor_parameters<executor_type>::type executor_parameters_type
The type of the associated executor parameters object which is associated with this execution
policy

typedef parallel_execution_tag execution_category
The category of the execution agents created by this execution policy.

Public Functions

parallel_unsequenced_policy operator() (task_policy_tag) const
Create a new parallel_unsequenced_policy from itself

Return The new parallel_unsequenced_policy

Parameters

• tag: [in] Specify that the corresponding asynchronous execution policy should be used

evaluator_type &executor()
Return the associated executor object.

evaluator_type const &executor() const
Return the associated evaluator object.

evaluator_parameters_type &parameters()
Return the associated evaluator parameters object.

evaluator_parameters_type const &parameters() const
Return the associated evaluator parameters object.
Private Functions

template <typename Archive>
void serialize (Archive &ar, const unsigned int version)

Private Members

executor_type exec_
executor_parameters_type params_

Friends

friend hpx::parallel::execution::hpx::serialization::access

struct sequenced_policy
#include <execution_policy.hpp> The class sequenced_policy is an execution policy type used
as a unique type to disambiguate parallel algorithm overloading and require that a parallel algo-

rithm’s execution may not be parallelized.

Subclassed by hpx::parallel::execution::sequenced_policy_shim< Executor, Parameters >

Public Types

typedef sequenced_executor executor_type
The type of the executor associated with this execution policy.

typedef execution::extract_executor_parameters<executor_type>::type executor_parameters_type
The type of the associated executor parameters object which is associated with this execution
policy

typedef sequenced_execution_tag execution_category
The category of the execution agents created by this execution policy.

Public Functions

sequenced_task_policy operator() (task_policy_tag) const
Create a new sequenced_task_policy.

Return The new sequenced_task_policy
Parameters
• tag: [in] Specify that the corresponding asynchronous execution policy should be used

template <typename Executor>
rebind_executor<sequenced_policy, Executor, executor_parameters_type>::type on (Executor
&&exec) const
Create a new sequenced_policy from the given executor

Note Requires: is_executor<Executor>::value is true

Return The new sequenced_policy

Template Parameters
• Executor: The type of the executor to associate with this execution policy.

Parameters
• exec: [in] The executor to use for the execution of the parallel algorithm the returned
execution policy is used with.

template <typename... Parameters, typename ParametersType = typename executor_parameters_join<Parameters...>::

rebind_executor<sequenced_policy, executor_type, ParametersType>::type with (Parameters&&... params)

Create a new sequenced_policy from the given execution parameters

Note Requires: all parameters are executor_parameters, different parameter types can’t be duplicated

Return The new sequenced_policy

Template Parameters
• Parameters: The type of the executor parameters to associate with this execution policy.

Parameters
• params: [in] The executor parameters to use for the execution of the parallel algorithm the returned execution policy is used with.

executor_type &executor ()
Return the associated executor object. Return the associated executor object.

executor_type const &executor () const
Return the associated executor object.

executor_parameters_type &parameters ()
Return the associated executor parameters object.

executor_parameters_type const &parameters () const
Return the associated executor parameters object.

Private Functions

template <typename Archive>
void serialize (Archive &ar, const unsigned int version)

Private Members

executor_type exec_

executor_parameters_type params_

Friends

friend hpx::parallel::execution::hpx::serialization::access

template <typename Executor_, typename Parameters_>
struct rebind
#include <execution_policy.hpp> Rebind the type of executor used by this execution policy. The execution category of Executor shall not be weaker than that of this execution policy

Public Types

typedef sequenced_policy_shim<Executor_, Parameters_> type
The type of the rebound execution policy.

struct sequenced_task_policy
#include <execution_policy.hpp> Extension: The class sequenced_task_policy is an execution policy type used as a unique type to disambiguate parallel algorithm overloading and indicate that a parallel algorithm’s execution may not be parallelized (has to run sequentially).
The algorithm returns a future representing the result of the corresponding algorithm when invoked with the `sequenced_policy`.

Subclassed by `hpx::parallel::execution::sequenced_task_policy_shim< Executor, Parameters >`

**Public Types**

```cpp
typedef sequenced_executor executor_type
```

The type of the executor associated with this execution policy.

```cpp
typedef execution::extract_executor_parameters<executor_type>::type executor_parameters_type
```

The type of the associated executor parameters object which is associated with this execution policy.

```cpp
typedef sequenced_execution_tag execution_category
```

The category of the execution agents created by this execution policy.

**Public Functions**

```cpp
sequenced_task_policy operator() (task_policy_tag) const
```

Create a new `sequenced_task_policy` from itself.

**Parameters**

- `tag`: [in] Specify that the corresponding asynchronous execution policy should be used

```cpp
template <typename Executor>
rebind_executor<sequenced_task_policy, Executor, executor_parameters_type>::type on (Executor &&exec) const
```

Create a new `sequenced_task_policy` from the given executor.

**Note** Requires: `is_executor<Executor>::value` is true

**Return** The new `sequenced_task_policy`

**Template Parameters**

- `Executor`: The type of the executor to associate with this execution policy.

```cpp
template <typename... Parameters, typename ParametersType = typename executor_parameters_join<Parameters...>::type>
rebind_executor<sequenced_task_policy, executor_type, ParametersType>::type with (Parameters&&... params) const
```

Create a new `sequenced_task_policy` from the given execution parameters.

**Note** Requires: all parameters are `executor_parameters`, different parameter types can’t be duplicated

**Return** The new `sequenced_task_policy`

**Template Parameters**

- `Parameters`: The type of the executor parameters to associate with this execution policy.

```cpp
executor_type &executor() const
```

Return the associated executor object.
executor_type const &executor() const
Return the associated executor object.

executor_parameters_type &parameters()  
Return the associated executor parameters object.

executor_parameters_type const &parameters() const  
Return the associated executor parameters object.

Private Functions

template <typename Archive>
void serialize(Archive &ar, const unsigned int version)

Private Members

executor_type exec_

executor_parameters_type params_

Friends

friend hpx::parallel::execution::hpx::serialization::access

template <typename Executor_, typename Parameters_>
struct rebind
#include <execution_policy.hpp>
Rebind the type of executor used by this execution policy. The execution category of Executor shall not be weaker than that of this execution policy.

Public Types

typedef sequenced_task_policy_shim<Executor_, Parameters_> type
The type of the rebound execution policy.

#include <hpx/parallel/executor_parameters.hpp>
#include <hpx/parallel/exception_list.hpp>
#include <hpx/parallel/executors/execution_information_fwd.hpp>
#include <hpx/parallel/executors/execution_parameters_fwd.hpp>
#include <hpx/parallel/executors/default_executor.hpp>
namespace hpx

namespace parallel

namespace execution

Typedefs

using hpx::parallel::execution::default_executor = typedef threads::executors::default_executor
Refers to the currently used base-executor.
#include <hpx/parallel/execution/service_executors.hpp>

namespace hpx

namespace parallel

namespace execution

Typedefs

using hpx::parallel::execution::service_executor = typedef threads::executors::service_executor

A service_executor exposes one of the predefined HPX thread pools through an executor interface.

Note All tasks executed by one of these executors will run on one of the OS-threads dedicated for the given thread pool. The tasks will not run as HPX-threads.

using hpx::parallel::execution::io_pool_executor = typedef threads::executors::io_pool_executor

A io_pool_executor exposes the predefined HPX IO thread pool through an executor interface.

Note All tasks executed by one of these executors will run on one of the OS-threads dedicated for the IO thread pool. The tasks will not run as HPX-threads.

using hpx::parallel::execution::parcel_pool_executor = typedef threads::executors::parcel_pool_executor

A io_pool_executor exposes the predefined HPX parcel thread pool through an executor interface.

Note All tasks executed by one of these executors will run on one of the OS-threads dedicated for the parcel thread pool. The tasks will not run as HPX-threads.

using hpx::parallel::execution::timer_pool_executor = typedef threads::executors::timer_pool_executor

A io_pool_executor exposes the predefined HPX timer thread pool through an executor interface.

Note All tasks executed by one of these executors will run on one of the OS-threads dedicated for the timer thread pool. The tasks will not run as HPX-threads.

using hpx::parallel::execution::main_pool_executor = typedef threads::executors::main_pool_executor

A io_pool_executor exposes the predefined HPX main thread pool through an executor interface.

Note All tasks executed by one of these executors will run on one of the OS-threads dedicated for the main thread pool. The tasks will not run as HPX-threads.

#include <hpx/parallel/execution/distribution_policy_executor.hpp>

namespace hpx

namespace parallel

namespace execution

Functions

template <typename DistPolicy>
distribution_policy_executor<typename hpx::util::decay<DistPolicy>::type> make_distribution_policy_executor(DistPolicy &&policy)

Create a new distribution_policy_executor from the given distribution_policy.
Parameters
  • policy: The distribution_policy to create an executor from

template <typename DistPolicy>
class distribution_policy_executor
#include <distribution_policy_executor.hpp> A distribution_policy_executor creates groups of parallel execution agents which execute in threads implicitly created by the executor and placed on any of the associated localities.

Template Parameters
  • DistPolicy: The distribution policy type for which an executor should be created. The expression `hpx::traits::is_distribution_policy<DistPolicy>::value` must evaluate to true.

Public Functions

template <typename DistPolicy_, typename Enable = typename std::enable_if< !std::is_same<
distribution_policy_executor
(DistPolicy_ &&policy)
Create a new distribution_policy executor from the given distribution policy

Parameters
  • policy: The distribution_policy to create an executor from

Private Members

DistPolicy policy_

#include <hpx/parallel/executors/dynamic_chunk_size.hpp>
namespace hpx

namespace parallel

namespace execution

struct dynamic_chunk_size
#include <dynamic_chunk_size.hpp> Loop iterations are divided into pieces of size chunk_size and then dynamically scheduled among the threads: when a thread finishes one chunk, it is dynamically assigned another. If chunk_size is not specified, the default chunk size is 1.

Note This executor parameters type is equivalent to OpenMP’s DYNAMIC scheduling directive.

Public Functions

dynamic_chunk_size (std::size_t chunk_size = 1)
Construct a dynamic_chunk_size executor parameters object

Parameters
  • chunk_size: [in] The optional chunk size to use as the number of loop iterations to schedule together. The default chunk size is 1.
#include <hpx/parallel/executors/auto_chunk_size.hpp>

namespace hpx

namespace parallel

namespace execution

struct auto_chunk_size
#include <auto_chunk_size.hpp>

Loop iterations are divided into pieces and then assigned to threads. The number of loop iterations combined is determined based on measurements of how long the execution of 1% of the overall number of iterations takes. This executor parameters type makes sure that as many loop iterations are combined as necessary to run for the amount of time specified.

Public Functions

auto_chunk_size()

Construct an auto_chunk_size executor parameters object

Note Default constructed auto_chunk_size executor parameter types will use 80 microseconds as the minimal time for which any of the scheduled chunks should run.

auto_chunk_size(hpx::util::steady_duration const &rel_time)

Construct an auto_chunk_size executor parameters object

Parameters

- rel_time: [in] The time duration to use as the minimum to decide how many loop iterations should be combined.

#include <hpx/parallel/executors/thread_timed_execution.hpp>

namespace hpx

namespace threads

Functions

template <typename Executor, typename F, typename... Ts>
std::enable_if<hpx::traits::is_threads_executor<Executor>::value>::type post_at (Executor &&exec,
hpx::util::steady_time_point const &abs_time,
F &&f, Ts&&... ts)

template <typename Executor, typename F, typename... Ts>

```cpp
std::enable_if<hpx::traits::is_threads_executor<Executor>::value>::type post_after(Executor &&exec,
    hpx::util::steady_duration const &rel_time,
    F &&f,
    Ts &&...,
    ts)

template <typename Executor, typename F, typename... Ts>
std::enable_if<hpx::traits::is_threads_executor<Executor>::value, hpx::future<typename hpx::util::detail::invoke_deferred_result<F, Ts...>::type>>::type async_execute_at(Executor &&exec,
    hpx::util::steady_time_point const &abs_time,
    F &&f,
    Ts &&...,
    ts)

template <typename Executor, typename F, typename... Ts>
std::enable_if<hpx::traits::is_threads_executor<Executor>::value, hpx::future<typename hpx::util::detail::invoke_deferred_result<F, Ts...>::type>>::type async_execute_after(Executor &&exec,
    hpx::util::steady_duration const &rel_time,
    F &&f,
    Ts &&...,
    ts)

template <typename Executor, typename F, typename... Ts>
std::enable_if<hpx::traits::is_threads_executor<Executor>::value, typename hpx::util::detail::invoke_deferred_result<F, Ts...>::type>::type sync_execute_at(Executor &&exec,
    hpx::util::steady_time_point const &abs_time,
    F &&f,
    Ts &&...,
    ts)

template <typename Executor, typename F, typename... Ts>
std::enable_if<hpx::traits::is_threads_executor<Executor>::value, typename hpx::util::detail::invoke_deferred_result<F, Ts...>::type>::type sync_execute_after(Executor &&exec,
    hpx::util::steady_duration const &rel_time,
    F &&f,
    Ts &&...,
    ts)
```

```cpp
#include <hpx/parallel/executors/execution_parameters.hpp>

namespace hpx
```

Chapter 2. What’s so special about HPX?
namespace parallel

namespace execution

Functions

template <typename... Params>
executor_parameters_join<Params...>::type join_executor_parameters(Params&&... params)

template <typename Param>
Param && join_executor_parameters (Param &&param)

template <typename... Params>
struct executor_parameters_join

Public Types

template<>
using type = detail::executor_parameters<
    typename hpx::util::decay<Params>::type...
>

template <typename Param>
template<>
struct executor_parameters_join<Param>

Public Types

template<>
using type = Param

#include <hpx/parallel/executors/thread_pool_attached_executors.hpp>

namespace hpx

namespace parallel

namespace execution

Typedefs

using hpx::parallel::execution::local_priority_queue_attached_executor = typedef threads::executors::local_priority_queue_attached_executor

#include <hpx/parallel/executors/thread_pool_os_executors.hpp>

namespace hpx

namespace parallel

namespace execution

2.9. API reference 493
Typedefs

`using hpx::parallel::execution::local_priority_queue_os_executor = typedef threads::executors::local_priority_queue_os_executor`

Creates a new local_priority_queue_executor

**Parameters**

- `max_punits`: [in] The maximum number of processing units to associate with the newly created executor.
- `min_punits`: [in] The minimum number of processing units to associate with the newly created executor (default: 1).

```cpp
#include <hpx/parallel/executors/guided_chunk_size.hpp>

namespace hpx

namespace parallel

namespace execution

struct guided_chunk_size
#include <guided_chunk_size.hpp> Iterations are dynamically assigned to threads in blocks as threads request them until no blocks remain to be assigned. Similar to dynamic_chunk_size except that the block size decreases each time a number of loop iterations is given to a thread. The size of the initial block is proportional to \( \frac{number\_of\_iterations}{number\_of\_cores} \). Subsequent blocks are proportional to \( \frac{number\_of\_iterations\_remaining}{number\_of\_cores} \). The optional chunk size parameter defines the minimum block size. The default chunk size is 1.

**Note** This executor parameters type is equivalent to OpenMP’s GUIDED scheduling directive.

**Public Functions**

`guided_chunk_size (std::size_t min_chunk_size = 1)`

Construct a guided_chunk_size executor parameters object

**Parameters**

- `min_chunk_size`: [in] The optional minimal chunk size to use as the minimal number of loop iterations to schedule together. The default minimal chunk size is 1.

```cpp
#include <hpx/parallel/executors/this_thread_executors.hpp>
#include <hpx/parallel/executors/fused_bulk_execute.hpp>
#include <hpx/parallel/executors/thread_pool_executors.hpp>

namespace hpx

namespace parallel

namespace execution

```
Typedefs

```cpp
using hpx::parallel::execution::local_priority_queue_executor = typedef threads::executors::local_priority_queue_executor
```

Creates a new `local_priority_queue_executor`

**Parameters**

- `max_punits`: [in] The maximum number of processing units to associate with the newly created executor.
- `min_punits`: [in] The minimum number of processing units to associate with the newly created executor (default: 1).

```cpp
#include <hpx/parallel/executors/timed_executors.hpp>

namespace hpx

namespace parallel

namespace execution

```cpp
Typedefs

```cpp
using hpx::parallel::execution::sequenced_timed_executor = typedef timed_executor<execution::sequenced_executor>
using hpx::parallel::execution::parallel_timed_executor = typedef timed_executor<execution::parallel_executor>
```

```cpp
template <typename BaseExecutor>
struct timed_executor
```

**Public Types**

```cpp
typedef std::decay<BaseExecutor>::type base_executor_type
typedef hpx::traits::executor_execution_category<base_executor_type>::type execution_category
typedef hpx::traits::executor_parameters_type<base_executor_type>::type parameters_type
```

**Public Functions**

```cpp
timed_executor (hpx::util::steady_time_point const &abs_time)

timed_executor (hpx::util::steady_duration const &rel_time)
```

```cpp
template <typename Executor>
timed_executor (Executor &&exec, hpx::util::steady_time_point const &abs_time)
```

```cpp
template <typename Executor>
timed_executor (Executor &&exec, hpx::util::steady_duration const &rel_time)
```

```cpp
template <typename F, typename... Ts>
hpx::util::detail::invoke_deferred_result<F, Ts...>::type sync_execute (F &&f, Ts&&... ts)
```

```cpp
template <typename F, typename... Ts>
hpx::future<typename hpx::util::detail::invoke_deferred_result<F, Ts...>::type> async_execute (F &&f, Ts&&... ts)
```

2.9. API reference
template <typename F, typename... Ts>
void post (F &&f, Ts&&... ts)

Public Members

BaseExecutor exec_
std::chrono::steady_clock::time_point execute_at_

#include <hpX/parallel/executors/rebind_executor.hpp>

namespace hpx
namespace parallel
namespace execution

template <typename ExecutionPolicy, typename Executor, typename Parameters>
struct rebind_executor
#include <rebind_executor.hpp>
Rebind the type of executor used by an execution policy. The execution category of Executor shall not be weaker than that of ExecutionPolicy.

Public Types

typedef ExecutionPolicy::template rebind<executor_type, parameters_type>::type type
The type of the rebound execution policy.

#include <hpX/parallel/executors/parallel_executor_aggregated.hpp>

template<>
template<>
struct parallel_policy_executor_aggregated<hpx::launch>

Public Types

template<>
using execution_category = parallel_execution_tag
Associate the parallel_execution_tag executor tag type as a default with this executor.

template<>
using executor_parameters_type = static_chunk_size
Associate the static_chunk_size executor parameters type as a default with this executor.

Public Functions

hpx::parallel::execution::parallel_policy_executor_aggregated::parallel_policy_executor_aggregated(hpx::launch l = hpx::launch::async_policy{}, std::size_t spread = 4, std::size_t tasks = std::size_t(-1))
Create a new parallel executor.

template <typename F, typename S, typename... Ts>
std::vector<hpx::future<void>> bulk_async_execute (F &&f, S const &shape, Ts&&... ts)
call const

namespace hpx
namespace parallel

namespace execution

typedefs

using hpx::parallel::execution::parallel_executor_aggregated = typedef parallel_policy_executor_aggregated

template <typename Policy = hpx::launch::async_policy>
struct parallel_policy_executor_aggregated
#include <parallel_executor_aggregated.hpp>

A parallel_executor_aggregated creates groups of parallel execution agents that execute in threads implicitly created by the executor. This executor prefers continuing with the creating thread first before executing newly created threads.

This executor conforms to the concepts of a TwoWayExecutor, and a BulkTwoWayExecutor

Public Types

template<>
using execution_category = parallel_execution_tag

Associate the parallel_execution_tag executor tag type as a default with this executor.

template<>
using executor_parameters_type = static_chunk_size

Associate the static_chunk_size executor parameters type as a default with this executor.

Public Functions

parallel_policy_executor_aggregated(\size_t spread = 4, \size_t tasks = \size_t(-1))

Create a new parallel executor.

template <typename F, typename S, typename... Ts>
std::vector<hpx::future<void>> bulk_async_execute(F &&f, S const &shape, Ts&&... ts) const

Public Types

template<>
using execution_category = parallel_execution_tag

Associate the parallel_execution_tag executor tag type as a default with this executor.

template<>
using executor_parameters_type = static_chunk_size

Associate the static_chunk_size executor parameters type as a default with this executor.

Public Functions

hpx::parallel::execution::parallel_policy_executor_aggregated::parallel_policy_executor_aggregated(hpx::launch l = hpx::launch::async_policy{}, std::size_t spread = 4, std::size_t tasks = std::size_t(-1))

Create a new parallel executor.

template <typename F, typename S, typename... Ts>

std::vector<hpx::future<void>> bulk_async_execute(F &&f, S const &shape, Ts&&... ts) const

#include <hpx/parallel/executors/thread_execution.hpp>

namespace hpx

namespace threads

Functions

template <typename Executor, typename F, typename... Ts>
std::enable_if<hpx::traits::is_threads_executor<Executor>::value, hpx::lcos::future<typename hpx::util::detail::invoke_deferred_result<F, Ts...>::type>>::type


template <typename Executor, typename F, typename... Ts>
std::enable_if<hpx::traits::is_threads_executor<Executor>::value, typename hpx::util::detail::invoke_deferred_result<F, Ts...>::type>::type


template <typename Executor, typename F, typename Future, typename... Ts>
std::enable_if<hpx::traits::is_threads_executor<Executor>::value, hpx::lcos::future<typename hpx::util::detail::invoke_deferred_result<F, Future, Ts...>::type>>::type


template <typename Executor, typename F, typename... Ts>
std::enable_if<hpx::traits::is_threads_executor<Executor>::value>::type


template <typename Executor, typename F, typename... Ts>
std::enable_if<hpx::traits::is_threads_executor<Executor>::value>::type

Chapter 2. What’s so special about HPX?
\begin{verbatim}

std::enable_if<hpx::traits::is_threads_executor<Executor>::value && std::is_same<typename hpx::util::decay<Hint>::type, hpx::threads::thread_schedule_hint>::value>::type
post(Executor&& exec, F&& f, Hint&& hint, const char* annotation, Ts&&... ts)

template <typename Executor, typename F, typename Shape, typename... Ts>
std::enable_if<hpx::traits::is_threads_executor<Executor>::value, std::vector<hpx::lcos::future<typename parallel::execution::detail::bulk_function_result<F, Shape, Ts...>::type>>>::type
bulk_async_execute(Executor&& exec, F&& f, Shape const& shape, Ts&&... ts)

template <typename Executor, typename F, typename Shape, typename... Ts>
std::enable_if<hpx::traits::is_threads_executor<Executor>::value, typename parallel::execution::detail::bulk_execute_result<F, Shape, Ts...>::type>::type
bulk_sync_execute(Executor&& exec, F&& f, Shape const& shape, Ts&&... ts)

template <typename Executor, typename F, typename Shape, typename Future, typename... Ts>
std::enable_if<hpx::traits::is_threads_executor<Executor>::value, hpx::future<typename parallel::execution::detail::bulk_then_execute_result<F, Shape, Future, Ts...>::type>>>::type
bulk_then_execute(Executor&& exec, F&& f, Shape const& shape, Future&& predecessor, Ts&&... ts)
\end{verbatim}
#include <hpx/parallel/executors/thread_pool_executor.hpp>

namespace hpx

    namespace parallel

        namespace execution

            struct thread_pool_executor
            #include <thread_pool_executor.hpp> A thread_pool_executor creates groups of parallel execution agents which execute in threads implicitly created by the executor. This executor prefers continuing with the creating thread first before executing newly created threads.

This executor conforms to the concepts of a TwoWayExecutor, and a BulkTwoWayExecutor

Public Types

typedef parallel_execution_tag execution_category
    Associate the parallel_execution_tag executor tag type as a default with this executor.

typedef static_chunk_size executor_parameters_type
    Associate the static_chunk_size executor parameters type as a default with this executor.

Public Functions

thread_pool_executor (threads::thread_pool_base *pool)
    Create a new parallel executor.

Private Members

threads::thread_pool_base *pool_

#include <hpx/parallel/executors/parallel_executor.hpp>

namespace hpx

    namespace parallel

        namespace execution

Typedefs

using hpx::parallel::execution::parallel_executor = typedef parallel_policy_executor<hpx::launch>

template <typename Policy>
struct parallel_policy_executor
    #include <parallel_executor.hpp> A parallel_executor creates groups of parallel execution agents which execute in threads implicitly created by the executor. This executor prefers continuing with the creating thread first before executing newly created threads.

This executor conforms to the concepts of a TwoWayExecutor, and a BulkTwoWayExecutor
Public Types

typedef parallel_execution_tag execution_category
Associate the parallel_execution_tag executor tag type as a default with this executor.

typedef static_chunk_size executor_parameters_type
Associate the static_chunk_size executor parameters type as a default with this executor.

Public Functions

parallel_policy_executor (Policy l = detail::get_default_policy<Policy>::call(), std::size_t spread = 4, std::size_t tasks = std::size_t(-1))
Create a new parallel executor.

#include <hpx/parallel/executors/persistent_auto_chunk_size.hpp>

namespace hpx

namespace parallel

namespace execution

struct persistent_auto_chunk_size
#include <persistent_auto_chunk_size.hpp> Loop iterations are divided into pieces and then assigned to threads. The number of loop iterations combined is determined based on measurements of how long the execution of 1% of the overall number of iterations takes. This executor parameters type makes sure that as many loop iterations are combined as necessary to run for the amount of time specified.

Public Functions

persistent_auto_chunk_size ()
Construct an persistent_auto_chunk_size executor parameters object

Note Default constructed persistent_auto_chunk_size executor parameter types will use 0 microseconds as the execution time for each chunk and 80 microseconds as the minimal time for which any of the scheduled chunks should run.

persistent_auto_chunk_size (hpx::util::steady_duration const &time_cs)
Construct an persistent_auto_chunk_size executor parameters object

Parameters

• time_cs: The execution time for each chunk.

persistent_auto_chunk_size (hpx::util::steady_duration const &time_cs,
 hpx::util::steady_duration const &rel_time)
Construct an persistent_auto_chunk_size executor parameters object

Parameters

• rel_time: [in] The time duration to use as the minimum to decide how many loop iterations should be combined.
• time_cs: The execution time for each chunk.
#include <hpx/parallel/executors/pool_executor.hpp>

namespace hpx

namespace parallel

namespace execution

Typedefs

using hpx::parallel::execution::pool_executor = typedef threads::executors::pool_executor

#include <hpx/parallel/executors/post_policy_dispatch.hpp>

#include <hpx/parallel/executors/sequenced_executor.hpp>

namespace hpx

namespace parallel

namespace execution

struct sequenced_executor

#include <sequenced_executor.hpp> A sequential_executor creates groups of sequential execution agents which execute in the calling thread. The sequential order is given by the lexicographical order of indices in the index space.

#include <hpx/parallel/executors/thread_execution_information.hpp>

namespace hpx

namespace threads

Functions

template <typename Executor, typename Parameters>
std::enable_if<hpx::traits::is_threads_executor<Executor>::value, std::size_t>::type processing_units_count(Executor&& exec, Parameters& params)

template <typename Executor>
std::enable_if<hpx::traits::is_threads_executor<Executor>::value, bool>::type has_pending_closures(Executor && exec)
```cpp
std::enable_if<hpx::traits::is_threads_executor<Executor>::value, threads::mask_cref_type>::type get_pu_mask (Executor &&exec, threads::topology &topo, std::size_t thread_num)

template <typename Executor, typename Mode>
std::enable_if<hpx::traits::is_threads_executor<Executor>::value>::type set_scheduler_mode (Executor &&exec, Mode mode)

#include <hpx/parallel/executors/execution_fwd.hpp>
namespace hpx
namespace parallel

namespace execution

struct parallel_execution_tag
#include <execution_fwd.hpp> Function invocations executed by a group of parallel execution agents execute in unordered fashion. Any such invocations executing in the same thread are indeterminately sequenced with respect to each other.

Note parallel_execution_tag is weaker than sequenced_execution_tag.

struct sequenced_execution_tag
#include <execution_fwd.hpp> Function invocations executed by a group of sequential execution agents execute in sequential order.

struct unsequenced_execution_tag
#include <execution_fwd.hpp> Function invocations executed by a group of vector execution agents are permitted to execute in unordered fashion when executed in different threads, and un-sequenced with respect to one another when executed in the same thread.

Note unsequenced_execution_tag is weaker than parallel_execution_tag.

#include <hpx/parallel/executors/static_chunk_size.hpp>
namespace hpx
namespace parallel

namespace execution

struct static_chunk_size
#include <static_chunk_size.hpp> Loop iterations are divided into pieces of size chunk_size and then assigned to threads. If chunk_size is not specified, the iterations are evenly (if possible) divided contiguously among the threads.

Note This executor parameters type is equivalent to OpenMP’s STATIC scheduling directive.
```
Public Functions

`static_chunk_size()`
Construct a static_chunk_size executor parameters object

**Note**  By default the number of loop iterations is determined from the number of available cores and the overall number of loop iterations to schedule.

`static_chunk_size(std::size_t chunk_size)`
Construct a static_chunk_size executor parameters object

**Parameters**
- `chunk_size`: [in] The optional chunk size to use as the number of loop iterations to run on a single thread.

```
#include <hpx/parallel/executors/execution_information.hpp>
namespace hpx

namespace threads

Functions

    threads::mask_cref_type get_pu_mask (threads::topology &topo, std::size_t thread_num)
```

```
#include <hpx/parallel/executors/timed_execution.hpp>
#include <hpx/parallel/traits/vector_pack_count_bits.hpp>
namespace hpx

namespace parallel

namespace traits

Functions

    HPX_HOST_DEVICE std::size_t hpx::parallel::traits::count_bits(bool value)
```

```
#include <hpx/parallel/traits/vector_pack_load_store.hpp>
#include <hpx/parallel/traits/vector_pack_type.hpp>
#include <hpx/parallel/traits/vector_pack_alignment_size.hpp>
#include <hpx/parallel/datapar/execution_policy_fwd.hpp>
#include <hpx/parallel/datapar/execution_policy.hpp>
```

```
#include <compatibility/hpx/util/histogram.hpp>
#include <compatibility/hpx/util/max.hpp>
```
#include <compatibility/hpx/util/min.hpp>
#include <compatibility/hpx/util/rolling_min.hpp>
#include <compatibility/hpx/util/rolling_max.hpp>
#include <hpx/statistics/histogram.hpp>

namespace boost
{

namespace accumulators
{

namespace extract
{

Variables

extractor<tag::histogram> const histogram = {};
}

namespace tag
{

struct histogram : public depends_on<count>, public histogram_num_bins, public histogram_min_range,

impl

template<typename Sample, typename Weight>
struct apply

Public Types

typedef hpx::util::detail::histogram_impl<Sample> type

#include <hpx/statistics/max.hpp>

namespace hpx
{

namespace util

Functions

template<typename T>
HPX_HOST_DEVICE T const&() hpx::util::max(T const & a, T const & b)

#include <hpx/statistics/min.hpp>

namespace hpx
{

namespace util

}
Functions

\[
\text{template <typename } T \text{>}
\]
\[
\text{HPX_HOST_DEVICE } T \text{ const}\&() \text{ hpx::util::min}(T \text{ const } & a, T \text{ const } & b)
\]

#include <hpx/statistics/rolling_min.hpp>

namespace boost

namespace accumulators

namespace extract

Variables

extractor<tag::rolling_min> const rolling_min = {};

namespace tag

struct rolling_min : public depends_on<rolling_window>

struct impl

\[
\text{template <typename } \text{Sample, typename } \text{Weight>}
\]

struct apply

Public Types

typedef hpx::util::detail::rolling_min_impl<Sample> type

#include <hpx/statistics/rolling_max.hpp>

namespace boost

namespace accumulators

namespace extract

Variables

extractor<tag::rolling_max> const rolling_max = {};

namespace tag

struct rolling_max : public depends_on<rolling_window>

struct impl
template <typename Sample, typename Weight>
struct apply

Public Types

typedef hpx::util::detail::rolling_max_impl<Sample> type

errors
#include <compatibility/hpx/throw_exception.hpp>
#include <compatibility/hpx/error.hpp>
#include <compatibility/hpx/error_code.hpp>
#include <compatibility/hpx/exception.hpp>
#include <compatibility/hpx/exception_fwd.hpp>
#include <compatibility/hpx/exception_info.hpp>
#include <compatibility/hpx/exception_list.hpp>
#include <hpx/errors/throw_exception.hpp>

Defines

HPX_THROW_EXCEPTION (errcode, f, msg)
Throw a hpx::exception initialized from the given parameters.

The macro HPX_THROW_EXCEPTION can be used to throw a hpx::exception. The purpose of this macro is to prepend the source file name and line number of the position where the exception is thrown to the error message. Moreover, this associates additional diagnostic information with the exception, such as file name and line number, locality id and thread id, and stack backtrace from the point where the exception was thrown.

The parameter errcode holds the hpx::error code the new exception should encapsulate. The parameter f is expected to hold the name of the function exception is thrown from and the parameter msg holds the error message the new exception should encapsulate.

void raise_exception()
{
    // Throw a hpx::exception initialized from the given parameters.
    // Additionally associate with this exception some detailed
    // diagnostic information about the throw-site.
    HPX_THROW_EXCEPTION(hpx::no_success, "raise_exception", "simulated error");
}

Example:

HPX_THROWS_IF (ec, errcode, f, msg)
Either throw a hpx::exception or initialize hpx::error_code from the given parameters.

The macro HPX_THROWS_IF can be used to either throw a hpx::exception or to initialize a hpx::error_code from the given parameters. If &ec == &hpx::throws, the semantics of this macro are equivalent to HPX_THROW_EXCEPTION. If &ec != &hpx::throws, the hpx::error_code instance ec is initialized instead.

The parameter errcode holds the hpx::error code from which the new exception should be initialized. The parameter f is expected to hold the name of the function exception is thrown from and the parameter msg holds
the error message the new exception should encapsulate.

```cpp
#include <hpx/errors/error.hpp>

namespace hpx
{

    Enums

    enum error
    {
        Possible error conditions.
        This enumeration lists all possible error conditions which can be reported from any of the API functions.
        Values:
        success = 0
            The operation was successful.
        no_success = 1
            The operation did failed, but not in an unexpected manner.
        not_implemented = 2
            The operation is not implemented.
        out_of_memory = 3
            The operation caused an out of memory condition.
        bad_action_code = 4
        bad_component_type = 5
            The specified component type is not known or otherwise invalid.
        network_error = 6
            A generic network error occurred.
        version_too_new = 7
            The version of the network representation for this object is too new.
        version_too_old = 8
            The version of the network representation for this object is too old.
        version_unknown = 9
            The version of the network representation for this object is unknown.
        unknown_component_address = 10
        duplicate_component_address = 11
            The given global id has already been registered.
        invalid_status = 12
            The operation was executed in an invalid status.
        bad_parameter = 13
            One of the supplied parameters is invalid.
        internal_server_error = 14
        service_unavailable = 15
        bad_request = 16
        repeated_request = 17
        lock_error = 18
    }
}
```
duplicate_console = 19
    There is more than one console locality.

no_registered_console = 20
    There is no registered console locality available.

startup_timed_out = 21

uninitialized_value = 22

bad_response_type = 23

deadlock = 24

assertion_failure = 25

null_thread_id = 26
    Attempt to invoke a API function from a non-HPX thread.

invalid_data = 27

yield_aborted = 28
    The yield operation was aborted.

dynamic_link_failure = 29

commandline_option_error = 30
    One of the options given on the command line is erroneous.

serialization_error = 31
    There was an error during serialization of this object.

unhandled_exception = 32
    An unhandled exception has been caught.

kernel_error = 33
    The OS kernel reported an error.

broken_task = 34
    The task associated with this future object is not available anymore.

task_moved = 35
    The task associated with this future object has been moved.

task_already_started = 36
    The task associated with this future object has already been started.

future_already_retrieved = 37
    The future object has already been retrieved.

promise_already_satisfied = 38
    The value for this future object has already been set.

future_does_not_support_cancellation = 39
    The future object does not support cancellation.

future_can_not_be_cancelled = 40
    The future can’t be canceled at this time.

no_state = 41
    The future object has no valid shared state.

broken.promise = 42
    The promise has been deleted.

thread_resource_error = 43
```
future_cancelled = 44
thread_cancelled = 45
thread_not_interruptable = 46
duplicate_component_id = 47
    The component type has already been registered.
unknown_error = 48
    An unknown error occurred.
bad_plugin_type = 49
    The specified plugin type is not known or otherwise invalid.
filesystem_error = 50
    The specified file does not exist or other filesystem related error.
bad_function_call = 51
    equivalent of std::bad_function_call
task_canceled_exception = 52
    parallel::v2::task_canceled_exception
task_block_not_active = 53
    task_region is not active
out_of_range = 54
    Equivalent to std::out_of_range.
length_error = 55
    Equivalent to std::length_error.
migration_needs_retry = 56
    migration failed because of global race, retry

#include <hpx/errors/error_code.hpp>

namespace hpx

    Unnamed Group

    error_code make_error_code (error e, throwmode mode = plain)
        Returns a new error_code constructed from the given parameters.
    error_code make_error_code (error e, char const *func, char const *file, long line, throwmode mode = plain)
    error_code make_error_code (error e, char const *msg, throwmode mode = plain)
        Returns error_code(e, msg, mode).
    error_code make_error_code (error e, char const *msg, char const *func, char const *file, long line, throwmode mode = plain)
    error_code make_error_code (error e, std::string const &msg, throwmode mode = plain)
        Returns error_code(e, msg, mode).
    error_code make_error_code (error e, std::string const &msg, char const *func, char const *file, long line, throwmode mode = plain)
    error_code make_error_code (std::exception_ptr const &e)
Functions

*boost::system::error_category const & get_hpx_category()*
Returns generic HPX error category used for new errors.

*boost::system::error_category const & get_hpx_rethrow_category()*
Returns generic HPX error category used for errors re-thrown after the exception has been de-serialized.

*error_code make_success_code(throwmode mode = plain)*
Returns *error_code(hpx::success, “success”, mode).*

**class error_code**

#include <error_code.hpp>
A *hpx::error_code* represents an arbitrary error condition.

The class *hpx::error_code* describes an object used to hold error code values, such as those originating from the operating system or other low-level application program interfaces.

**Note** Class *hpx::error_code* is an adjunct to error reporting by exception

**Public Functions**

*error_code (throwmode mode = plain)*
Construct an object of type *error_code*.

**Parameters**
- *mode*: The parameter *mode* specifies whether the constructed *hpx::error_code* belongs to the error category *hpx_category* (if mode is *plain*, this is the default) or to the category *hpx_category_rethrow* (if mode is *rethrow*).

**Exceptions**
- *nothing:*

*error_code (error e, throwmode mode = plain)*
Construct an object of type *error_code*.

**Parameters**
- *e*: The parameter *e* holds the *hpx::error* code the new exception should encapsulate.
- *mode*: The parameter *mode* specifies whether the constructed *hpx::error_code* belongs to the error category *hpx_category* (if mode is *plain*, this is the default) or to the category *hpx_category_rethrow* (if mode is *rethrow*).

**Exceptions**
- *nothing:*

*error_code (error e, char const *func, char const *file, long line, throwmode mode = plain)*
Construct an object of type *error_code*.

**Parameters**
- *e*: The parameter *e* holds the *hpx::error* code the new exception should encapsulate.
- *func*: The name of the function where the error was raised.
- *file*: The file name of the code where the error was raised.
- *line*: The line number of the code line where the error was raised.
- *mode*: The parameter *mode* specifies whether the constructed *hpx::error_code* belongs to the error category *hpx_category* (if mode is *plain*, this is the default) or to the category *hpx_category_rethrow* (if mode is *rethrow*).

**Exceptions**
- *nothing:*

*error_code (error e, char const *msg, throwmode mode = plain)*
Construct an object of type *error_code*.
Parameters
- \( e \): The parameter \( e \) holds the \texttt{hpx::error} code the new exception should encapsulate.
- \( \text{msg} \): The parameter \( \text{msg} \) holds the error message the new exception should encapsulate.
- \( \text{mode} \): The parameter \( \text{mode} \) specifies whether the constructed \texttt{hpx::error_code} belongs to the error category \texttt{hpx_category} (if mode is \texttt{plain}, this is the default) or to the category \texttt{hpx_category_rethrow} (if mode is \texttt{rethrow}).

Exceptions
- \texttt{std::bad_alloc}: (if allocation of a copy of the passed string fails).

\texttt{error\_code (error e, char const *msg, char const *func, char const *file, long line, throwmode mode = plain)}

Construct an object of type \texttt{error\_code}.

Parameters
- \( e \): The parameter \( e \) holds the \texttt{hpx::error} code the new exception should encapsulate.
- \( \text{msg} \): The parameter \( \text{msg} \) holds the error message the new exception should encapsulate.
- \( \text{func} \): The name of the function where the error was raised.
- \( \text{file} \): The file name of the code where the error was raised.
- \( \text{line} \): The line number of the code line where the error was raised.
- \( \text{mode} \): The parameter \( \text{mode} \) specifies whether the constructed \texttt{hpx::error_code} belongs to the error category \texttt{hpx_category} (if mode is \texttt{plain}, this is the default) or to the category \texttt{hpx_category_rethrow} (if mode is \texttt{rethrow}).

Exceptions
- \texttt{std::bad_alloc}: (if allocation of a copy of the passed string fails).

\texttt{error\_code (error e, std::string const &msg, throwmode mode = plain)}

Construct an object of type \texttt{error\_code}.

Parameters
- \( e \): The parameter \( e \) holds the \texttt{hpx::error} code the new exception should encapsulate.
- \( \text{msg} \): The parameter \( \text{msg} \) holds the error message the new exception should encapsulate.
- \( \text{func} \): The name of the function where the error was raised.
- \( \text{file} \): The file name of the code where the error was raised.
- \( \text{line} \): The line number of the code line where the error was raised.
- \( \text{mode} \): The parameter \( \text{mode} \) specifies whether the constructed \texttt{hpx::error_code} belongs to the error category \texttt{hpx_category} (if mode is \texttt{plain}, this is the default) or to the category \texttt{hpx_category_rethrow} (if mode is \texttt{rethrow}).

Exceptions
- \texttt{std::bad_alloc}: (if allocation of a copy of the passed string fails).

\texttt{std::string get\_message () const}

Return a reference to the error message stored in the \texttt{hpx::error\_code}.

Exceptions
• nothing:

    void clear()
    Clear this error_code object. The postconditions of invoking this method are.
    • value() == hpx::success and category() == hpx::get_hpx_category()

    error_code(error_code const & rhs)
    Copy constructor for error_code

    Note This function maintains the error category of the left hand side if the right hand side is a success
code.

    error_code & operator=(error_code const & rhs)
    Assignment operator for error_code

    Note This function maintains the error category of the left hand side if the right hand side is a success
code.

Private Functions

    error_code (int err, hpx::exception const & e)
    error_code (std::exception_ptr const & e)

Private Members

    std::exception_ptr exception_

Friends

    friend hpx::exception
    error_code make_error_code(std::exception_ptr const & e)

#include <hpx/errors/exception.hpp>
namespace hpx

Typedefs

    using hpx::custom_exception_info_handler_type = typedef std::function<hpx::exception_info(std::string const&,
                                                                                           std::string const&,
                                                                                           long, std::string const&)>  
    using hpx::pre_exception_handler_type = typedef std::function<void()>

Functions

    void set_custom_exception_info_handler(custom_exception_info_handler_type f)
    void set_pre_exception_handler(pre_exception_handler_type f)

    std::string get_error_what(exception_info const & xi)
    Return the error message of the thrown exception.

    The function hpx::get_error_what can be used to extract the diagnostic information element representing
the error message as stored in the given exception instance.
Return The error message stored in the exception. If the exception instance does not hold this information, the function will return an empty string.

See hpx::diagnostic_information(), hpx::get_error_host_name(), hpx::get_error_process_id(), hpx::get_error_function_name(), hpx::get_error_file_name(), hpx::get_error_line_number(), hpx::get_error_os_thread(), hpx::get_error_thread_id(), hpx::get_error_thread_description(), hpx::get_error() hpx::get_error_backtrace(), hpx::get_error_env(), hpx::get_error_config(), hpx::get_error_state()

Parameters

• xi: The parameter e will be inspected for the requested diagnostic information elements which have been stored at the point where the exception was thrown. This parameter can be one of the following types: hpx::exception_info, hpx::error_code, std::exception, or std::exception_ptr.

Exceptions

• std::bad_alloc: (if one of the required allocations fails)

```cpp
error get_error (hpx::exception const &e)
```

Return the error code value of the exception thrown.

The function hpx::get_error can be used to extract the diagnostic information element representing the error value code as stored in the given exception instance.

Return The error value code of the locality where the exception was thrown. If the exception instance does not hold this information, the function will return hpx::naming::invalid_locality_id.

See hpx::diagnostic_information(), hpx::get_error_host_name(), hpx::get_error_process_id(), hpx::get_error_function_name(), hpx::get_error_file_name(), hpx::get_error_line_number(), hpx::get_error_os_thread(), hpx::get_error_thread_id(), hpx::get_error_thread_description(), hpx::get_error_backtrace(), hpx::get_error_env(), hpx::get_error_what(), hpx::get_error_config(), hpx::get_error_state()

Parameters

• e: The parameter e will be inspected for the requested diagnostic information elements which have been stored at the point where the exception was thrown. This parameter can be one of the following types: hpx::exception, hpx::error_code, or std::exception_ptr.

Exceptions

• nothing:

```cpp
error get_error (hpx::error_code const &e)
```

```cpp
std::string get_error_function_name (hpx::exception_info const &xi)
```

Return the function name from which the exception was thrown.

The function hpx::get_error_function_name can be used to extract the diagnostic information element representing the name of the function as stored in the given exception instance.

Return The name of the function from which the exception was thrown. If the exception instance does not hold this information, the function will return an empty string.

See hpx::diagnostic_information(), hpx::get_error_host_name(), hpx::get_error_process_id(), hpx::get_error_file_name(), hpx::get_error_line_number(), hpx::get_error_os_thread(), hpx::get_error_thread_id(), hpx::get_error_thread_description(), hpx::get_error_backtrace(), hpx::get_error_env(), hpx::get_error_what(), hpx::get_error_config(), hpx::get_error_state()
Parameters

• \( xi \): The parameter \( e \) will be inspected for the requested diagnostic information elements which have been stored at the point where the exception was thrown. This parameter can be one of the following types: hpx::exception_info, hpx::error_code, std::exception, or std::exception_ptr.

Exceptions

• std::bad_alloc: (if one of the required allocations fails)

`std::string get_error_file_name(hpx::exception_info const &xi)`

Return the (source code) file name of the function from which the exception was thrown.

The function hpx::get_error_file_name can be used to extract the diagnostic information element representing the name of the source file as stored in the given exception instance.

Return The name of the source file of the function from which the exception was thrown. If the exception instance does not hold this information, the function will return an empty string.

See hpx::diagnostic_information(), hpx::get_error_host_name(), hpx::get_error_process_id(), hpx::get_error_function_name(), hpx::get_error_line_number(), hpx::get_error_os_thread(), hpx::get_error_thread_id(), hpx::get_error_thread_description(), hpx::get_error(), hpx::get_error_backtrace(), hpx::get_error_env(), hpx::get_error_what(), hpx::get_error_config(), hpx::get_error_state()

Parameters

• \( xi \): The parameter \( e \) will be inspected for the requested diagnostic information elements which have been stored at the point where the exception was thrown. This parameter can be one of the following types: hpx::exception_info, hpx::error_code, std::exception, or std::exception_ptr.

Exceptions

• std::bad_alloc: (if one of the required allocations fails)

`long get_error_line_number(hpx::exception_info const &xi)`

Return the line number in the (source code) file of the function from which the exception was thrown.

The function hpx::get_error_line_number can be used to extract the diagnostic information element representing the line number as stored in the given exception instance.

Return The line number of the place where the exception was thrown. If the exception instance does not hold this information, the function will return -1.

See hpx::diagnostic_information(), hpx::get_error_host_name(), hpx::get_error_process_id(), hpx::get_error_function_name(), hpx::get_error_line_number(), hpx::get_error_os_thread(), hpx::get_error_thread_id(), hpx::get_error_thread_description(), hpx::get_error(), hpx::get_error_backtrace(), hpx::get_error_env(), hpx::get_error_what(), hpx::get_error_config(), hpx::get_error_state()
class exception: public system_error
#include <exception.hpp> A hpx::exception is the main exception type used by HPX to report errors.

The hpx::exception type is the main exception type used by HPX to report errors. Any exceptions thrown by functions in the HPX library are either of this type or of a type derived from it. This implies that it is always safe to use this type only in catch statements guarding HPX library calls.

Subclassed by hpx::exception_list

Public Functions

exception (error e = success)  
Construct a hpx::exception from a hpx::error.

Parameters  
• e: The parameter e holds the hpx::error code the new exception should encapsulate.

exception (boost::system::system_error const &e)  
Construct a hpx::exception from a boost::system_error.

exception (boost::system::error_code const &e)  
Construct a hpx::exception from a boost::system::error_code (this is new for Boost V1.69). This constructor is required to compensate for the changes introduced as a resolution to LWG3162 (https://cplusplus.github.io/LWG/issue3162).

exception (error e, char const *msg, throwmode mode = plain)  
Construct a hpx::exception from a hpx::error and an error message.

Parameters  
• e: The parameter e holds the hpx::error code the new exception should encapsulate.
• msg: The parameter msg holds the error message the new exception should encapsulate.
• mode: The parameter mode specifies whether the returned hpx::error_code belongs to the error category hpx_category (if mode is plain, this is the default) or to the category hpx_category_rethrow (if mode is rethrow).

exception (error e, std::string const &msg, throwmode mode = plain)  
Construct a hpx::exception from a hpx::error and an error message.

Parameters  
• e: The parameter e holds the hpx::error code the new exception should encapsulate.
• msg: The parameter msg holds the error message the new exception should encapsulate.
• mode: The parameter mode specifies whether the returned hpx::error_code belongs to the error category hpx_category (if mode is plain, this is the default) or to the category hpx_category_rethrow (if mode is rethrow).

~exception()  
Destruct a hpx::exception

Exceptions  
• nothing:

error get_error() const  
The function get_error() returns the hpx::error code stored in the referenced instance of a hpx::exception. It returns the hpx::error code this exception instance was constructed from.

Exceptions  
• nothing:
error_code get_error_code(throwmode mode = plain) const

The function get_error_code() returns a hpx::error_code which represents the same error condition as this hpx::exception instance.

Parameters
- mode: The parameter mode specifies whether the returned hpx::error_code belongs to the error category hpx_category (if mode is plain, this is the default) or to the category hpx_category_rethrow (if mode is rethrow).

struct thread_interrupted: public exception

#include <exception.hpp> A hpx::thread_interrupted is the exception type used by HPX to interrupt a running HPX thread.

The hpx::thread_interrupted type is the exception type used by HPX to interrupt a running thread.

A running thread can be interrupted by invoking the interrupt() member function of the corresponding hpx::thread object. When the interrupted thread next executes one of the specified interruption points (or if it is currently blocked whilst executing one) with interruption enabled, then a hpx::thread_interrupted exception will be thrown in the interrupted thread. If not caught, this will cause the execution of the interrupted thread to terminate. As with any other exception, the stack will be unwound, and destructors for objects of automatic storage duration will be executed.

If a thread wishes to avoid being interrupted, it can create an instance of hpx::this_thread::disable_interruption. Objects of this class disable interruption for the thread that created them on construction, and restore the interruption state to whatever it was before on destruction.

```cpp
void f()
{
    // interruption enabled here
    {
        hpx::this_thread::disable_interruption di;
        // interruption disabled
        {
            hpx::this_thread::disable_interruption di2;
            // interruption still disabled
        } // di2 destroyed, interruption state restored
        // interruption still disabled
    } // di destroyed, interruption state restored
    // interruption now enabled
}
```

The effects of an instance of hpx::this_thread::disable_interruption can be temporarily reversed by constructing an instance of hpx::this_thread::restore_interruption, passing in the hpx::this_thread::disable_interruption object in question. This will restore the interruption state to what it was when the hpx::this_thread::disable_interruption object was constructed, and then disable interruption again when the hpx::this_thread::restore_interruption object is destroyed.

```cpp
void g()
{
    // interruption enabled here
    {
        hpx::this_thread::disable_interruption di;
        // interruption disabled
        {
            hpx::this_thread::restore_interruption ri(di);
            // interruption now enabled
        } // ri destroyed, interruption disable again
        // di destroyed, interruption state restored
    }
}
```

(continues on next page)
At any point, the interruption state for the current thread can be queried by calling `hpx::this_thread::interruption_enabled()`.

```cpp
#include <hpx/errors/exception_fwd.hpp>

namespace hpx
{

  // interruption now enabled
}
```

### Enums

**enum throwmode**

Encode error category for new `error_code`.

Values:

- `plain` = 0
- `rethrow` = 1

### Variables

**error_code throws**

Predefined `error_code` object used as “throw on error” tag.

The predefined `hpx::error_code` object `hpx::throws` is supplied for use as a “throw on error” tag.

Functions that specify an argument in the form `error_code & ec=throws` (with appropriate namespace qualifiers), have the following error handling semantics:

- If `&ec != &throws` and an error occurred: `ec.value()` returns the implementation specific error number for the particular error that occurred and `ec.category()` returns the error_category for `ec.value()`.
- If `&ec != &throws` and an error did not occur, `ec.clear()`.
- If an error occurs and `&ec == &throws`, the function throws an exception of type `hpx::exception` or of a type derived from it. The exception’s `get_errorcode()` member function returns a reference to an `hpx::error_code` object with the behavior as specified above.

```cpp
#include <hpx/errors/exception_info.hpp>

#define HPX_DEFINE_ERROR_INFO(NAME, TYPE)

namespace hpx
{

  template <typename E>
  HPX_NORETURN void hpx::throw_with_info(E && e, exception_info && xi = exception_info)
  {

  template <typename E>
  HPX_NORETURN void hpx::throw_with_info(E && e, exception_info const & xi)
  {

  template <typename E>
  ```
exception_info *get_exception_info (E &e)

template <typename E>
exception_info const *get_exception_info (E const &e)

template <typename E, typename F>
auto invoke_with_exception_info (E const &e, F &&f)

template <typename F>
auto invoke_with_exception_info (std::exception_ptr const &p, F &&f)

template <typename F>
auto invoke_with_exception_info (hpx::error_code const &ec, F &&f)

template <typename Tag, typename Type>
struct error_info

Public Types

template<>
using tag = Tag

template<>
using type = Type

Public Functions

error_info (Type const &value)
error_info (Type &&value)

Public Members

Type _value

class exception_info
Subclassed by hpx::detail::exception_with_info_base

Public Functions

exception_info ()
exception_info (exception_info const &other)
exception_info (exception_info &&other)

exception_info &operator= (exception_info const &other)
exception_info &operator= (exception_info &&other)

virtual ~exception_info ()

template <typename... ErrorInfo>
exception_info &set (ErrorInfo&&... tagged_values)

template <typename Tag>
Tag::type const *get () const
**Private Types**

using node_ptr = std::shared_ptr<detail::exception_info_node_base>

**Private Members**

node_ptr _data

#include <hpx/errors/exception_list.hpp>

namespace hpx

class exception_list : public hpx::exception
#include <exception_list.hpp> The class exception_list is a container of exception_ptr objects parallel algorithms may use to communicate uncaught exceptions encountered during parallel execution to the caller of the algorithm.

The type exception_list::const_iterator fulfills the requirements of a forward iterator.

**Public Types**

typedef exception_list_type::const_iterator iterator

bidirectional iterator

**Public Functions**

std::size_t size() const

The number of exception_ptr objects contained within the exception_list.

Note Complexity: Constant time.

dependent_exception_list_type::const_iterator begin() const

An iterator referring to the first exception_ptr object contained within the exception_list.

dependent_exception_list_type::const_iterator end() const

An iterator which is the past-the-end value for the exception_list.

**Private Types**

typedef boost::detail::spinlock mutex_type

typedef std::list<std::exception_ptr> exception_list_type

**Private Members**

exception_list_type exceptions_

mutex_type mtx_

type_support
#include <compatibility/hpx/util/unwrap_ref.hpp>

#include <compatibility/hpx/util/lazy_conditional.hpp>

#include <compatibility/hpx/util/always_void.hpp>
```cpp
#include <compatibility/hpx/util/unused.hpp>
#include <compatibility/hpx/util/decay.hpp>
#include <compatibility/hpx/util/identity.hpp>
#include <compatibility/hpx/util/detected.hpp>
#include <compatibility/hpx/util/static.hpp>
#include <compatibility/hpx/util/void_guard.hpp>
#include <compatibility/hpx/util/lazy_enable_if.hpp>
#include <hpx/type_support/unwrap_ref.hpp>

template <typename T>
template<>
struct unwrap_reference<boost::reference_wrapper<T>>

   Public Types

   typedef T type


```

```cpp
namespace hpx
namespace util

Functions

   template <typename T>
   unwrap_reference<T>::type &unwrap_ref(T &t)
```


```cpp
template <typename T>
struct unwrap_reference

    Public Types

typedef T type

template <typename T>
    template<>
struct unwrap_reference<boost::reference_wrapper<T>>

    Public Types

typedef T type

template <typename T>
    template<>
struct unwrap_reference<boost::reference_wrapper<T> const>

    Public Types

typedef T type

template <typename T>
    template<>
struct unwrap_reference<std::reference_wrapper<T>>

    Public Types

typedef T type

template <typename T>
    template<>
struct unwrap_reference<std::reference_wrapper<T> const>

    Public Types

typedef T type

#include <hpx/type_support/lazy_conditional.hpp>

#include <hpx/type_support/always_void.hpp>

namespace hpx

namespace util

    template <typename... T>
struct always_void

    Public Types

typedef void type
```
#include <hpx/type_support/unused.hpp>

Defines

HPX_UNUSED(x)

namespace hpx

namespace util

Variables

HPX_CONSTEXPR_OR_CONST unused_type hpx::util::unused = unused_type()

struct unused_type

Public Functions

HPX_HOST_DEVICE unused_type()

HPX_HOST_DEVICE unused_type(unused_type const&)

HPX_HOST_DEVICE unused_type(unused_type&&)

template<typename T>
HPX_HOST_DEVICE unused_type(T const&)

template<typename T>
HPX_HOST_DEVICE unused_type const& hpx::util::unused_type::operator=(T const &)

HPX_HOST_DEVICE unused_type& hpx::util::unused_type::operator=(T const &)

HPX_HOST_DEVICE unused_type const& hpx::util::unused_type::operator=(unused_type const &)

HPX_HOST_DEVICE unused_type& hpx::util::unused_type::operator=(unused_type const &)

HPX_HOST_DEVICE unused_type& hpx::util::unused_type::operator=(unused_type &&)

HPX_HOST_DEVICE unused_type const& hpx::util::unused_type::operator=(unused_type &&)

#include <hpx/type_support/decay.hpp>

#include <hpx/type_support/identity.hpp>

namespace hpx

namespace util

template<typename T>
struct identity

Public Types

typedef T type
#include <hpx/type_support/detected.hpp>

namespace hpx

namespace util

Typedefs

using hpx::util::is_detected = typedef typename detail::detector<nonesuch, void, Op, Args...>::value_t
using hpx::util::detected_t = typedef typename detail::detector<nonesuch, void, Op, Args...>::type
using hpx::util::detected_or = typedef detail::detector<Default, void, Op, Args...>
using hpx::util::detected_or_t = typedef typename detected_or<Default, Op, Args...>::type
using hpx::util::is_detected_exact = typedef std::is_same<Expected, detected_t<Op, Args...> >
using hpx::util::is_detected_convertible = typedef std::is_convertible<detected_t<Op, Args...>, To>

struct nonesuch

Public Functions

nonesuch()
~nonesuch()

nonesuch(nonesuch const&)

void operator=(nonesuch const&)

#include <hpx/type_support/static.hpp>

Defines

HPX_EXPORT_STATIC_

namespace hpx

namespace util

template <typename T, typename Tag = T>
struct static_

Public Types

typedef T value_type
typedef T & reference
typedef T const & const_reference
Public Functions

HPX_NON_COPYABLE (static_)

static_()

operator reference()

operator const_reference() const

reference get()

const_reference get() const

Private Types

typedef std::add_pointer<value_type>::type pointer

typedef std::aligned_storage<sizeof(value_type), std::alignment_of<value_type>::value>::type storage_type

Private Static Functions

static pointer get_address()

Private Static Attributes

static_<T, Tag>::storage_type data_

std::once_flag constructed_

struct default_constructor

Public Static Functions

template<>

static void construct()

struct destructor

Public Functions

template<>

~destructor()

#include <hpx/type_support/void_guard.hpp>

namespace hpx

namespace util

    template <>
    template<>
    struct void_guard<void>
Public Functions

```cpp
template <typename T>
HPX_HOST_DEVICE void hpx::util::void_guard::operator,(T const &) const
```

```cpp
#include <hpx/type_support/lazy_enable_if.hpp>

namespace hpx

namespace util

```cpp
template <typename T>
template<class>
struct lazy_enable_if<true, T>
```

Public Types

```cpp
typedef T::type type
```

```cpp
cache
#include <compatibility/hpx/util/cache/local_cache.hpp>

#include <compatibility/hpx/util/cache/lru_cache.hpp>

#include <compatibility/hpx/util/cache/statistics/local_statistics.hpp>

#include <compatibility/hpx/util/cache/statistics/no_statistics.hpp>

#include <compatibility/hpx/util/cache/statistics/local_full_statistics.hpp>

#include <compatibility/hpx/util/cache/entries/entry.hpp>

#include <compatibility/hpx/util/cache/entries/size_entry.hpp>

#include <compatibility/hpx/util/cache/entries/lfu_entry.hpp>

#include <compatibility/hpx/util/cache/entries/fifo_entry.hpp>

#include <compatibility/hpx/util/cache/entries/lru_entry.hpp>

#include <compatibility/hpx/util/cache/policies/always.hpp>

#include <hpx/cache/local_cache.hpp>

namespace hpx

namespace util

namespace cache

```cpp
template <typename Key, typename Entry, typename UpdatePolicy = std::less<Entry>, typename InsertPolicy = policies::always<Entry>, typename CacheStorage = std::map<Key, Entry>, typename Statistics = statistics::no_statistics>
```
class local_cache
#include <hpx/cache/local_cache.hpp> The local_cache implements the basic functionality needed for a local (non-distributed) cache.

Template Parameters
- **Key**: The type of the keys to use to identify the entries stored in the cache
- **Entry**: The type of the items to be held in the cache, must model the CacheEntry concept
- **UpdatePolicy**: A (optional) type specifying a (binary) function object used to sort the cache entries based on their ‘age’. The ‘oldest’ entries (according to this sorting criteria) will be discarded first if the maximum capacity of the cache is reached. The default is std::less<Entry>. The function object will be invoked using 2 entry instances of the type Entry. This type must model the UpdatePolicy model.
- **InsertPolicy**: A (optional) type specifying a (unary) function object used to allow global decisions whether a particular entry should be added to the cache or not. The default is policies::always, imposing no global insert related criteria on the cache. The function object will be invoked using the entry instance to be inserted into the cache. This type must model the InsertPolicy model.
- **CacheStorage**: A (optional) container type used to store the cache items. The container must be an associative and STL compatible container. The default is a std::map<Key, Entry>.
- **Statistics**: A (optional) type allowing to collect some basic statistics about the operation of the cache instance. The type must conform to the CacheStatistics concept. The default value is the type statistics::no_statistics which does not collect any numbers, but provides empty stubs allowing the code to compile.

Public Types

typedef Key key_type
typedef Entry entry_type
typedef UpdatePolicy update_policy_type
typedef InsertPolicy insert_policy_type
typedef CacheStorage storage_type
typedef Statistics statistics_type
typedef entry_type::value_type value_type
typedef storage_type::size_type size_type
typedef storage_type::value_type storage_value_type

Public Functions

local_cache(size_type max_size = 0, update_policy_type const &up = update_policy_type(), insert_policy_type const &ip = insert_policy_type())
Construct an instance of a local_cache.

Parameters
- **max_size**: [in] The maximal size this cache is allowed to reach any time. The default is zero (no size limitation). The unit of this value is usually determined by the unit of the values returned by the entry’s get_size function.
- **up**: [in] An instance of the UpdatePolicy to use for this cache. The default is to use a default constructed instance of the type as defined by the UpdatePolicy template parameter.
- **ip**: [in] An instance of the InsertPolicy to use for this cache. The default is to use a default constructed instance of the type as defined by the InsertPolicy template parameter.
local_cache (local_cache &&other)

size_type size() const
   Return current size of the cache.

Return The current size of this cache instance.

size_type capacity() const
   Access the maximum size the cache is allowed to grow to.

Note The unit of this value is usually determined by the unit of the return values of the entry’s function entry::get_size.

Return The maximum size this cache instance is currently allowed to reach. If this number is zero the cache has no limitation with regard to a maximum size.

bool reserve (size_type max_size)
   Change the maximum size this cache can grow to.

Return This function returns true if successful. It returns false if the new max_size is smaller than the current limit and the cache could not be shrunk to the new maximum size.

Parameters
   • max_size: [in] The new maximum size this cache will be allowed to grow to.

bool holds_key (key_type const &k) const
   Check whether the cache currently holds an entry identified by the given key.

Note This function does not call the entry’s function entry::touch. It just checks if the cache contains an entry corresponding to the given key.

Return This function returns true if the cache holds the referenced entry, otherwise it returns false.

Parameters
   • k: [in] The key for the entry which should be looked up in the cache.

bool get_entry (key_type const &k, key_type &realkey, entry_type &val)
   Get a specific entry identified by the given key.

Note The function will call the entry’s entry::touch function if the value corresponding to the provided key is found in the cache.

Return This function returns true if the cache holds the referenced entry, otherwise it returns false.

Parameters
   • k: [in] The key for the entry which should be retrieved from the cache.
   • val: [out] If the entry indexed by the key is found in the cache this value on successful return will be a copy of the corresponding entry.

bool get_entry (key_type const &k, entry_type &val)
   Get a specific entry identified by the given key.

Note The function will call the entry’s entry::touch function if the value corresponding to the provided key is found in the cache.

Return This function returns true if the cache holds the referenced entry, otherwise it returns false.

Parameters
   • k: [in] The key for the entry which should be retrieved from the cache.

bool get_entry (key_type const &k, value_type &val)
   Get a specific entry identified by the given key.
**Note** The function will call the entry’s `entry::touch` function if the value corresponding to the provided is found in the cache.

**Return** This function returns `true` if the cache holds the referenced entry, otherwise it returns `false`.

**Parameters**
- `k`: [in] The key for the entry which should be retrieved from the cache
- `val`: [out] If the entry indexed by the key is found in the cache this value on successful return will be a copy of the corresponding value.

```cpp
bool insert (key_type const &k, value_type const &val)
```
Insert a new element into this cache.

**Note** This function invokes both, the insert policy as provided to the constructor and the function `entry::insert` of the newly constructed entry instance. If either of these functions returns false the key/value pair doesn’t get inserted into the cache and the `insert` function will return `false`. Other reasons for this function to fail (return `false`) are a) the key/value pair is already held in the cache or b) inserting the new value into the cache maxed out its capacity and it was not possible to free any of the existing entries.

**Return** This function returns `true` if the entry has been successfully added to the cache, otherwise it returns `false`.

**Parameters**
- `k`: [in] The key for the entry which should be added to the cache.
- `value`: [in] The value which should be added to the cache.

```cpp
bool insert (key_type const &k, entry_type &e)
```
Insert a new entry into this cache.

**Note** This function invokes both, the insert policy as provided to the constructor and the function `entry::insert` of the provided entry instance. If either of these functions returns false the key/value pair doesn’t get inserted into the cache and the `insert` function will return `false`. Other reasons for this function to fail (return `false`) are a) the key/value pair is already held in the cache or b) inserting the new value into the cache maxed out its capacity and it was not possible to free any of the existing entries.

**Return** This function returns `true` if the entry has been successfully added to the cache, otherwise it returns `false`.

**Parameters**
- `k`: [in] The key for the entry which should be added to the cache.
- `value`: [in] The entry which should be added to the cache.

```cpp
bool update (key_type const &k, value_type const &val)
```
Update an existing element in this cache.

**Note** The function will call the entry’s `entry::touch` function if the indexed value is found in the cache.

**Note** The difference to the other overload of the `insert` function is that this overload replaces the cached value only, while the other overload replaces the whole cache entry, updating the cache entry properties.

**Return** This function returns `true` if the entry has been successfully updated, otherwise it returns `false`. If the entry currently is not held by the cache it is added and the return value reflects the outcome of the corresponding insert operation.

**Parameters**
- `k`: [in] The key for the value which should be updated in the cache.
- `value`: [in] The value which should be used as a replacement for the existing value in the cache. Any existing cache entry is not changed except for its value.

```cpp
template <typename F>
```
bool update_if (key_type const &k, value_type const &val, F f)
Update an existing element in this cache.

Note The function will call the entry’s entry::touch function if the indexed value is found in
the cache.

Note The difference to the other overload of the insert function is that this overload replaces
the cached value only, while the other overload replaces the whole cache entry, updating the
cache entry properties.

Return This function returns true if the entry has been successfully updated, otherwise it
returns false. If the entry currently is not held by the cache it is added and the return value
reflects the outcome of the corresponding insert operation.

Parameters
• k: [in] The key for the value which should be updated in the cache.
• value: [in] The value which should be used as a replacement for the existing value in
the cache. Any existing cache entry is not changed except for its value.
• f: [in] A callable taking two arguments, k and the key found in the cache (in that order).
  If f returns true, then the update will continue. If f returns false, then the update will not
  succeed.

bool update (key_type const &k, entry_type &e)
Update an existing entry in this cache.

Note The function will call the entry’s entry::touch function if the indexed value is found in
the cache.

Note The difference to the other overload of the insert function is that this overload replaces
the whole cache entry, while the other overload replaces the cached value only, leaving the
cache entry properties untouched.

Return This function returns true if the entry has been successfully updated, otherwise it
returns false. If the entry currently is not held by the cache it is added and the return value
reflects the outcome of the corresponding insert operation.

Parameters
• k: [in] The key for the entry which should be updated in the cache.
• value: [in] The entry which should be used as a replacement for the existing entry in the
  cache. Any existing entry is first removed and then this entry is added.

template <typename Func>
size_type erase (Func const &ep = policies::always<storage_value_type>()
Remove stored entries from the cache for which the supplied function object returns true.

Return This function returns the overall size of the removed entries (which is the sum of the
values returned by the entry::get_size functions of the removed entries).

Parameters
• ep: [in] This parameter has to be a (unary) function object. It is invoked for each of the
  entries currently held in the cache. An entry is considered for removal from the cache
  whenever the value returned from this invocation is true. Even then the entry might not be
  removed from the cache as its entry::remove function might return false.

size_type erase ()
Remove all stored entries from the cache.

Note All entries are considered for removal, but in the end an entry might not be removed from
the cache as its entry::remove function might return false. This function is very useful for
instance in conjunction with an entry’s entry::remove function enforcing additional criteria
like entry expiration, etc.

Return This function returns the overall size of the removed entries (which is the sum of the
values returned by the entry::get_size functions of the removed entries).
void clear()
    Clear the cache.
    Unconditionally removes all stored entries from the cache.

statistics_type const &get_statistics() const
    Allow to access the embedded statistics instance.

    Return  This function returns a reference to the statistics instance embedded inside this cache

statistics_type &get_statistics()

Protected Functions

bool free_space(long num_free)

Private Types

typedef storage_type::iterator iterator

typedef storage_type::const_iterator const_iterator

typedef std::deque<iterator> heap_type

typedef heap_type::iterator heap_iterator

typedef adapt<UpdatePolicy, iterator> adapted_update_policy_type

typedef statistics_type::update_on_exit update_on_exit

Private Members

size_type max_size_

size_type current_size_

storage_type store_

heap_type entry_heap_

adapted_update_policy_type update_policy_

insert_policy_type insert_policy_

statistics_type statistics_

template <typename Func, typename Iterator>
struct adapt

Public Functions

template<> adapt(Func f)

template<> bool operator() (Iterator const &lhs, Iterator const &rhs) const

Public Members

template<> Func f_
The lru_cache implements the basic functionality needed for a local (non-distributed) LRU cache.

**Template Parameters**
- **Key**: The type of the keys to use to identify the entries stored in the cache
- **Entry**: The type of the items to be held in the cache.
- **Statistics**: A (optional) type allowing to collect some basic statistics about the operation of the cache instance. The type must conform to the CacheStatistics concept. The default value is the type statistics::no_statistics which does not collect any numbers, but provides empty stubs allowing the code to compile.

**Public Types**

```cpp
typedef Key key_type
typedef Entry entry_type
typedef Statistics statistics_type
typedef std::pair<key_type, entry_type> entry_pair
typedef std::list<entry_pair> storage_type
typedef std::map<Key, typename storage_type::iterator> map_type
typedef std::size_t size_type
```

**Public Functions**

```cpp
lru_cache (size_type max_size = 0)
Construct an instance of a lru_cache.

Parameters
- **max_size**: [in] The maximal size this cache is allowed to reach any time. The default is zero (no size limitation). The unit of this value is usually determined by the unit of the values returned by the entry’s get_size function.

lru_cache (lru_cache &&other)

size_type size () const
Return The current size of the cache.

Return The current size of this cache instance.

size_type capacity () const
Access the maximum size the cache is allowed to grow to.
```
**Note** The unit of this value is usually determined by the unit of the return values of the entry’s function `entry::get_size`.

**Return** The maximum size this cache instance is currently allowed to reach. If this number is zero the cache has no limitation with regard to a maximum size.

```cpp
void reserve(size_type max_size)
```

Change the maximum size this cache can grow to.

**Parameters**
- `max_size`: [in] The new maximum size this cache will be allowed to grow to.

```cpp
bool holds_key(key_type const &key)
```

Check whether the cache currently holds an entry identified by the given key.

**Note** This function does not call the entry’s function `entry::touch`. It just checks if the cache contains an entry corresponding to the given key.

**Return** This function returns `true` if the cache holds the referenced entry, otherwise it returns `false`.

**Parameters**
- `k`: [in] The key for the entry which should be looked up in the cache.

```cpp
bool get_entry(key_type const &key, entry_type &entry)
```

Get a specific entry identified by the given key.

**Note** The function will “touch” the entry and mark it as recently used if the key was found in the cache.

**Return** This function returns `true` if the cache holds the referenced entry, otherwise it returns `false`.

**Parameters**
- `key`: [in] The key for the entry which should be retrieved from the cache.
- `entry`: [out] If the entry indexed by the key is found in the cache this value on successful return will be a copy of the corresponding entry.

```cpp
bool get_entry(key_type const &key, entry_type &entry)
```

Get a specific entry identified by the given key.

**Note** The function will “touch” the entry and mark it as recently used if the key was found in the cache.

**Return** This function returns `true` if the cache holds the referenced entry, otherwise it returns `false`.

**Parameters**
- `key`: [in] The key for the entry which should be retrieved from the cache.
- `entry`: [out] If the entry indexed by the key is found in the cache this value on successful return will be a copy of the corresponding entry.

```cpp
bool insert(key_type const &key, entry_type const &entry)
```

Insert a new entry into this cache.

**Note** This function assumes that the entry is not in the cache already. Inserting an already existing entry is considered undefined behavior

**Parameters**
- `key`: [in] The key for the entry which should be added to the cache.
- `entry`: [in] The entry which should be added to the cache.

```cpp
void insert_nonexist(key_type const &key, entry_type const &entry)
```

```cpp
void update(key_type const &key, entry_type const &entry)
```

Update an existing element in this cache.
**Note** The function will “touch” the entry and mark it as recently used if the key was found in the cache.

**Note** The difference to the other overload of the `insert` function is that this overload replaces the cached value only, while the other overload replaces the whole cache entry, updating the cache entry properties.

**Parameters**
- `key`: [in] The key for the value which should be updated in the cache.
- `entry`: [in] The entry which should be used as a replacement for the existing value in the cache. Any existing cache entry is not changed except for its value.

```cpp
template <typename F>
bool update_if (key_type const &key, entry_type const &entry, F &&f)
```
Update an existing element in this cache.

**Note** The function will “touch” the entry and mark it as recently used if the key was found in the cache.

**Note** The difference to the other overload of the `insert` function is that this overload replaces the cached value only, while the other overload replaces the whole cache entry, updating the cache entry properties.

**Return** This function returns `true` if the entry has been successfully updated, otherwise it returns `false`. If the entry currently is not held by the cache it is added and the return value reflects the outcome of the corresponding insert operation.

**Parameters**
- `key`: [in] The key for the value which should be updated in the cache.
- `entry`: [in] The value which should be used as a replacement for the existing value in the cache. Any existing cache entry is not changed except for its value.
- `f`: [in] A callable taking two arguments, `k` and the key found in the cache (in that order). If `f` returns true, then the update will continue. If `f` returns false, then the update will not succeed.

```cpp
template <typename Func>
size_type erase (Func const &ep)
```
Remove stored entries from the cache for which the supplied function object returns true.

**Return** This function returns the overall size of the removed entries (which is the sum of the values returned by the `entry::get_size` functions of the removed entries).

**Parameters**
- `ep`: [in] This parameter has to be a (unary) function object. It is invoked for each of the entries currently held in the cache. An entry is considered for removal from the cache whenever the value returned from this invocation is `true`.

```cpp
size_type erase ()
```
Remove all stored entries from the cache.

**Return** This function returns the overall size of the removed entries (which is the sum of the values returned by the `entry::get_size` functions of the removed entries).

```cpp
size_type clear ()
```
Clear the cache. Unconditionally removes all stored entries from the cache.

**statistics_type const &get_statistics () const**
Allow to access the embedded statistics instance.

**Return** This function returns a reference to the statistics instance embedded inside this cache.
**Private Types**

typedef statistics_type::update_on_exit update_on_exit

**Private Functions**

void touch(typename storage_type::iterator it)

void evict()

**Private Members**

size_type max_size_

size_type current_size_

storage_type storage_

map_type map_

statistics_type statistics_

#include <hpx/cache/statistics/local_statistics.hpp>

namespace hpx

namespace util

namespace cache

namespace statistics

class local_statistics : public hpx::util::cache::statistics::no_statistics

Subclassed by hpx::util::cache::statistics::local_full_statistics

**Public Functions**

local_statistics()

std::size_t get_and_reset (std::size_t &value, bool reset)

std::size_t hits () const

std::size_t misses () const

std::size_t insertions () const

std::size_t evictions () const

std::size_t hits (bool reset)

std::size_t misses (bool reset)

std::size_t insertions (bool reset)
std::size_t evictions(bool reset)

void got_hit()
    The function got_hit will be called by a cache instance whenever a entry got touched.

void got_miss()
    The function got_miss will be called by a cache instance whenever a requested entry has not
    been found in the cache.

void got_insertion()
    The function got_insertion will be called by a cache instance whenever a new entry has been
    inserted.

void got_eviction()
    The function got_eviction will be called by a cache instance whenever an entry has been
    removed from the cache because a new inserted entry let the cache grow beyond its capacity.

void clear()
    Reset all statistics.

**Private Members**

std::size_t hits_
std::size_t misses_
std::size_t insertions_
std::size_t evictions_

#include <hpx/cache/statistics/no_statistics.hpp>

namespace hpx
    namespace util
        namespace cache
            namespace statistics

    Enums

enum method
    Values:
    method_get_entry = 0
    method_insert_entry = 1
    method_update_entry = 2
    method_erase_entry = 3

class no_statistics
    Subclassed by hpx::util::cache::statistics::local_statistics
Public Functions

void got_hit ()
    The function got_hit will be called by a cache instance whenever a entry got touched.

void got_miss ()
    The function got_miss will be called by a cache instance whenever a requested entry has not been found in the cache.

void got_insertion ()
    The function got_insertion will be called by a cache instance whenever a new entry has been inserted.

void got_eviction ()
    The function got_eviction will be called by a cache instance whenever an entry has been removed from the cache because a new inserted entry let the cache grow beyond its capacity.

void clear ()
    Reset all statistics.

std::int64_t get_get_entry_count (bool)
    The function get_get_entry_count returns the number of invocations of the get_entry() API function of the cache.

std::int64_t get_insert_entry_count (bool)
    The function get_insert_entry_count returns the number of invocations of the insert_entry() API function of the cache.

std::int64_t get_update_entry_count (bool)
    The function get_update_entry_count returns the number of invocations of the update_entry() API function of the cache.

std::int64_t get_erase_entry_count (bool)
    The function get_erase_entry_count returns the number of invocations of the erase() API function of the cache.

std::int64_t get_get_entry_time (bool)
    The function get_get_entry_time returns the overall time spent executing of the get_entry() API function of the cache.

std::int64_t get_insert_entry_time (bool)
    The function get_insert_entry_time returns the overall time spent executing of the insert_entry() API function of the cache.

std::int64_t get_update_entry_time (bool)
    The function get_update_entry_time returns the overall time spent executing of the update_entry() API function of the cache.

std::int64_t get_erase_entry_time (bool)
    The function get_erase_entry_time returns the overall time spent executing of the erase() API function of the cache.

struct update_on_exit
    #include <no_statistics.hpp> Helper class to update timings and counts on function exit.

Public Functions

update_on_exit (no_statistics const& method)
# HPX Documentation, 1.4.1

```cpp
#include <hpx/cache/statistics/local_full_statistics.hpp>

namespace hpx
{
    namespace util
    {
        namespace cache
        {
            namespace statistics
            {

                class local_full_statistics : public hpx::util::cache::statistics::local_statistics
                {
                    // Public Functions
                    std::int64_t get_get_entry_count (bool reset)
                    {
                        return get_get_entry_count_;
                    }

                    std::int64_t get_insert_entry_count (bool reset)
                    {
                        return get_insert_entry_count_;
                    }

                    std::int64_t get_update_entry_count (bool reset)
                    {
                        return get_update_entry_count_;
                    }

                    std::int64_t get_erase_entry_count (bool reset)
                    {
                        return get_erase_entry_count_;
                    }

                    std::int64_t get_get_entry_time (bool reset)
                    {
                        return get_get_entry_time_;
                    }

                    std::int64_t get_insert_entry_time (bool reset)
                    {
                        return get_insert_entry_time_;
                    }

                    std::int64_t get_update_entry_time (bool reset)
                    {
                        return get_update_entry_time_;
                    }

                    std::int64_t get_erase_entry_time (bool reset)
                    {
                        return get_erase_entry_time_;
                    }

                    // Private Functions
                    std::int64_t get_and_reset_value (std::int64_t &value, bool reset)
                    {
                        return get_and_reset_value_;
                    }
                }
            }
        }
    }
}
```
Private Members

api_counter_data get_entry_
api_counter_data insert_entry_
api_counter_data update_entry_
api_counter_data erase_entry_

Friends

friend hpx::util::cache::statistics::update_on_exit

struct api_counter_data

Public Functions

api_counter_data() 

Public Members

std::int64_t count_
std::int64_t time_

struct update_on_exit
#include <local_full_statistics.hpp> Helper class to update timings and counts on function exit.

Public Functions

update_on_exit (local_full_statistics &stat, method m)
~update_on_exit() 

Public Members

std::int64_t started_at_
api_counter_data &data_

Private Static Functions

static api_counter_data &get_api_counter_data (local_full_statistics &stat, method m)

static std::uint64_t now() 

#include <hpx/cache/entries/entry.hpp>

namespace hpx

namespace util
namespace cache

namespace entries

template <typename Value, typename Derived>
class entry : boost::less_than_comparable<detail::derived<Value, Derived>::type>
#include <hpx/cache/entries/entry.hpp>

Template Parameters

• Value: The data type to be stored in a cache. It has to be default constructible, copy constructible and less_than_comparable.
• Derived: The (optional) type for which this type is used as a base class.

Public Types

typedef Value value_type

Public Functions

t entry ()
Any cache entry has to be default constructible.

entry (value_type const &val)
Construct a new instance of a cache entry holding the given value.

bool touch ()
The function touch is called by a cache holding this instance whenever it has been requested (touched).

Note It is possible to change the entry in a way influencing the sort criteria mandated by the UpdatePolicy. In this case the function should return true to indicate this to the cache, forcing to reorder the cache entries.

Note This function is part of the CacheEntry concept

Return This function should return true if the cache needs to update it’s internal heap. Usually this is needed if the entry has been changed by touch() in a way influencing the sort order as mandated by the cache’s UpdatePolicy

bool insert ()
The function insert is called by a cache whenever it is about to be inserted into the cache.

Note This function is part of the CacheEntry concept

Return This function should return true if the entry should be added to the cache, otherwise it should return false.

bool remove ()
The function remove is called by a cache holding this instance whenever it is about to be removed from the cache.

Note This function is part of the CacheEntry concept

Return The return value can be used to avoid removing this instance from the cache. If the value is true it is ok to remove the entry, other wise it will stay in the cache.

std::size_t get_size () const
Return the ‘size’ of this entry. By default the size of each entry is just one (1), which is sensible if the cache has a limit (capacity) measured in number of entries.
value_type &get ()
    Get a reference to the stored data value.

    Note This function is part of the CacheEntry concept

value_type const &get () const

Private Members

value_type value_

Friends

bool operator< (entry const &lhs, entry const &rhs)
    Forwarding operator< allowing to compare entries in stead of the values.

#include <hpx/cache/entries/size_entry.hpp>
namespace hpx

    namespace util

        namespace cache

            namespace entries

        template <typename Value, typename Derived>
        class size_entry : public hpx::util::cache::entries::entry<Value, detail::size_derived<Value, Derived>::type>

        #include <hpx/cache/entries/size_entry.hpp>

    The size_entry type can be used to store values in a cache which have a size associated (such as files, etc.). Using this type as the cache’s entry type makes sure that the entries with the biggest size are discarded from the cache first.

    Note The size_entry conforms to the CacheEntry concept.
    Note This type can be used to model a ‘discard smallest first’ cache policy if it is used with a std::greater as the caches’ UpdatePolicy (instead of the default std::less).

Template Parameters

- Value: The data type to be stored in a cache. It has to be default constructible, copy constructible and less_than_comparable.
- Derived: The (optional) type for which this type is used as a base class.

Public Functions

size_entry ()
    Any cache entry has to be default constructible.

size_entry (Value const &val, std::size_t size)
    Construct a new instance of a cache entry holding the given value.

std::size_t get_size () const
    Return the ‘size’ of this entry.
**Private Types**

typedef detail::size_derived<Value, Derived>::type derived_type
typedef entry<Value, derived_type> base_type

**Private Members**

std::size_t size_

**Friends**

bool operator<(size_entry const &lhs, size_entry const &rhs)

- Compare the ‘age’ of two entries. An entry is ‘older’ than another entry if it has a bigger size.

#include <hpx/cache/entries/lfu_entry.hpp>

namespace hpx

namespace util

namespace cache

namespace entries

template<typename Value>
class lfu_entry : public hpx::util::cache::entries::entry<Value, lfu_entry<Value>>

- The lfu_entry type can be used to store arbitrary values in a cache. Using this type as the cache’s entry type makes sure that the least frequently used entries are discarded from the cache first.

**Note** The lfu_entry conforms to the CacheEntry concept.

**Note** This type can be used to model a ‘most frequently used’ cache policy if it is used with a std::greater as the caches’ UpdatePolicy (instead of the default std::less).

**Template Parameters**

- Value: The data type to be stored in a cache. It has to be default constructible, copy constructible and less_than_comparable.

**Public Functions**

lfu_entry()

- Any cache entry has to be default constructible.

lfu_entry(Value const &val)

- Construct a new instance of a cache entry holding the given value.

bool touch()

- The function touch is called by a cache holding this instance whenever it has been requested (touched).
In the case of the LFU entry we store the reference count tracking the number of times this entry has been requested. This which will be used to compare the age of an entry during the invocation of the `operator<()`.

**Return** This function should return true if the cache needs to update it’s internal heap. Usually this is needed if the entry has been changed by `touch()` in a way influencing the sort order as mandated by the cache’s UpdatePolicy.

```cpp
unsigned long const &get_access_count() const
```

**Private Types**

typedef `entry<Value, lfu_entry<Value>>` base_type

**Private Members**

```cpp
unsigned long ref_count_
```

**Friends**

```cpp
bool operator< (lfu_entry const &lhs, lfu_entry const &rhs)
```

Compare the ‘age’ of two entries. An entry is ‘older’ than another entry if it has been accessed less frequently (LFU).

```cpp
#include <hpx/cache/entries/fifo_entry.hpp>
namespace hpx
{
    namespace util
    {
        namespace cache
        {
            namespace entries
            {
                template<typename Value>
                class fifo_entry: public hpx::util::cache::entries::entry<Value, fifo_entry<Value>>
                {
                    #include <hpx/cache/entries/fifo_entry.hpp>
                    The fifo_entry type can be used to store arbitrary values in a cache. Using this type as the cache’s entry type makes sure that the least recently inserted entries are discarded from the cache first.

                    **Note** The fifo_entry conforms to the CacheEntry concept.
                    **Note** This type can be used to model a ‘last in first out’ cache policy if it is used with a std::greater as the caches’ UpdatePolicy (instead of the default std::less).

                    **Template Parameters**
                    • Value: The data type to be stored in a cache. It has to be default constructible, copy constructible and less_than_comparable.

                    **Public Functions**

                    ```cpp
                    fifo_entry()
                    ```
                    Any cache entry has to be default constructible.
**fifo_entry** (Value const &val)
Construct a new instance of a cache entry holding the given value.

bool insert ()
The function insert is called by a cache whenever it is about to be inserted into the cache.

**Note**  This function is part of the CacheEntry concept

**Return**  This function should return true if the entry should be added to the cache, otherwise it should return false.

`std::chrono::steady_clock::time_point const & get_creation_time ()` const

**Private Types**

typedef entry<Value, fifo_entry<Value>> base_type

**Private Members**

`std::chrono::steady_clock::time_point insertion_time_`

**Friends**

bool operator< (fifo_entry const &lhs, fifo_entry const &rhs)
Compare the ‘age’ of two entries. An entry is ‘older’ than another entry if it has been created earlier (FIFO).

#include <hpx/cache/entries/lru_entry.hpp>

namespace hpx

namespace util

namespace cache

namespace entries


**template** <typename Value>

**class lru_entry : public hpx::util::cache::entries::entry<Value, lru_entry<Value>>**

#include <hpx/cache/entries/lru_entry.hpp>  The lru_entry type can be used to store arbitrary values in a cache. Using this type as the cache’s entry type makes sure that the least recently used entries are discarded from the cache first.

**Note**  The lru_entry conforms to the CacheEntry concept.

**Note**  This type can be used to model a ‘most recently used’ cache policy if it is used with a `std::greater` as the caches’ UpdatePolicy (instead of the default `std::less`).

**Template Parameters**

- **Value**: The data type to be stored in a cache. It has to be default constructible, copy constructible and `less_than_comparable`. 
Public Functions

`lru_entry()`
Any cache entry has to be default constructible.

`lru_entry(Value const &val)`
Construct a new instance of a cache entry holding the given value.

`bool touch()`
The function `touch` is called by a cache holding this instance whenever it has been requested (touched).

In the case of the LRU entry we store the time of the last access which will be used to compare the age of an entry during the invocation of the `operator<()`.

**Return** This function should return true if the cache needs to update it’s internal heap. Usually this is needed if the entry has been changed by `touch()` in a way influencing the sort order as mandated by the cache’s UpdatePolicy.

`std::chrono::steady_clock::time_point get_access_time() const`
Returns the last access time of the entry.

Private Types

`typedef entry<Value, lru_entry<Value>> base_type`

Private Members

`std::chrono::steady_clock::time_point access_time_`

Friends

`bool operator<(lru_entry const &lhs, lru_entry const &rhs)`
Compare the ‘age’ of two entries. An entry is ‘older’ than another entry if it has been accessed less recently (LRU).

```cpp
#include <hpx/cache/policies/always.hpp>

namespace hpx

namespace util

namespace cache

namespace policies

template <typename Entry>
struct always

Public Functions

`bool operator()(Entry const&)`
assertion
#include <compatibility/hpx/util/assert.hpp>
#include <hpx/assertion.hpp>

Defines

HPX_ASSERT(expr)
This macro asserts that expr evaluates to true.
If expr evaluates to false, the source location and msg is being printed along with the expression and additional. Afterwards the program is being aborted. The assertion handler can be costumized by calling hpx::assertion::set_assertion_handler().

Parameters
• expr: The expression to assert on. This can either be an expression that’s convertible to bool or a callable which returns bool
• msg: The optional message that is used to give further information if the assert fails. This should be convertible to a std::string

Asserts are enabled if HPX_DEBUG is set. This is the default for CMAKE_BUILD_TYPE=Debug

HPX_ASSERT_MSG(expr, msg)
HPX_ASSERT

namespace hpx

namespace assertion

Typedefs

using hpx::assertion::assertion_handler = typedef void (*)(source_location const& loc, const char* expr, std::string const& msg)
The signature for an assertion handler.

Functions

void set_assertion_handler(assertion_handler handler)
Set the assertion handler to be used within a program. If the handler has been set already once, the call to this function will be ignored.
Note This function is not thread safe

#include <hpx/assertion/evaluate_assert.hpp>
#include <hpx/assertion/current_function.hpp>

Defines

HPX_ASSERT_CURRENT_FUNCTION

#include <hpx/assertion/force_linking.hpp>

namespace hpx

namespace assertion
Functions

\texttt{force\_linking\_helper} \& \texttt{force\_linking}()

\textbf{struct force\_linking\_helper}

\textbf{Public Members}

\begin{verbatim}
void (*handle_assert)(source_location const&, const char*, std::string const&)
\end{verbatim}

\#include <hpx/assertion/source_location.hpp>

\begin{verbatim}
namespace hpx

namespace assertion

Functions

std::ostream &operator<<(std::ostream &os, source_location const &loc)

\end{verbatim}

\textbf{struct source\_location}

\begin{verbatim}
\#include <source\_location.hpp> This contains the location information where \texttt{HPX\_ASSERT} has been called

\end{verbatim}

\textbf{Public Members}

\begin{verbatim}
const char *file\_name
unsigned line\_number
const char *function\_name
\end{verbatim}

\textbf{resource\_partitioner}

\#include <compatibility/hpx/runtime/resource/partitioner\_fwd.hpp>

\#include <compatibility/hpx/runtime/resource/partitioner.hpp>

\#include <hpx/resource\_partitioner/partitioner\_fwd.hpp>

\begin{verbatim}
namespace hpx

namespace resource

\textbf{Typedefs}

using hpx::resource::scheduler\_function = typedef util::function\_nonser<std::unique_ptr<hpx::threads::thread\_pool\_base>>(hpx::threads::thread\_pool\_init\_parameters, hpx::threads::policies::thread\_queue\_init\_parameters)

\textbf{Enums}

\textbf{enum partitioner\_mode}

This enumeration describes the modes available when creating a resource partitioner.

\texttt{Values}:
mode_default = 0
    Default mode.

mode_allow_oversubscription = 1
    Allow processing units to be oversubscribed, i.e. multiple worker threads to share a single processing unit.

mode_allow_dynamic_pools = 2
    Allow worker threads to be added and removed from thread pools.

enum scheduling_policy
    This enumeration lists the available scheduling policies (or schedulers) when creating thread pools.
    Values:
    user_defined = -2
    unspecified = -1
    local = 0
    local_priority_fifo = 1
    local_priority_lifo = 2
    static_ = 3
    static_priority = 4
    abp_priority_fifo = 5
    abp_priority_lifo = 6
    shared_priority = 7

Functions

detail::partitioner &get_partitioner()
    May be used anywhere in code and returns a reference to the single, global resource partitioner.

bool is_partitioner_valid()
    Returns true if the resource partitioner has been initialized. Returns false otherwise.

#include <hpx/resource_partitioner/partitioner.hpp>

namespace hpx

namespace resource

class core

    Public Functions

    core (std::size_t id = invalid_core_id, numa_domain *domain = nullptr)
    std::vector<pu> const &pus() const
    std::size_t id() const
Private Functions

```
std::vector<core> cores_sharing_numa_domain()
```

Private Members

```
std::size_t id_
numa_domain *domain_
std::vector<pu> pus_
```

Private Static Attributes

```
std::size_t(-1)
```

Friends

```
friend hpx::resource::pu
friend hpx::resource::numa_domain
```

class numa_domain

Public Functions

```
uma_domain(std::size_t id = invalid_numa_domain_id)
std::vector<core> const &cores() const
std::size_t id() const
```

Private Members

```
std::size_t id_
std::vector<core> cores_
```

Private Static Attributes

```
std::size_t(-1)
```

Friends

```
friend hpx::resource::pu
friend hpx::resource::core
```

class partitioner

Public Functions

```
partitioner(util::function_nonser<int> hpx::program_options::variables_map &vm
             > const &f, hpx::program_options::options_description const &desc_cmdline, int argc,
char **argv, std::vector<std::string> ini_config, resource::partitioner_mode rpmode = resource::mode_default,
runtime_mode mode = runtime_mode_default
```

2.9. API reference
partitioner (util::function_nonser<int> int, char **
> const &f, int argc, char **argv, resource::partitioner_mode rpmode = resource::mode_default, hpx::runtime_mode mode = hpx::runtime_mode_default

partitioner (util::function_nonser<int> int, char **
> const &f, int argc, char **argv, std::vector<std::string> const &cfg, resource::partitioner_mode rpmode = resource::mode_default, hpx::runtime_mode mode = hpx::runtime_mode_default

partitioner (int argc, char **argv, resource::partitioner_mode rpmode = resource::mode_default, runtime_mode mode = runtime_mode_default)

partitioner (int argc, char **argv, std::vector<std::string> ini_config, resource::partitioner_mode rpmode = resource::mode_default, runtime_mode mode = runtime_mode_default)

partitioner (hpx::program_options::options_description const &desc_cmdline, int argc, char **argv, resource::partitioner_mode rpmode = resource::mode_default, runtime_mode mode = runtime_mode_default)

partitioner (hpx::program_options::options_description const &desc_cmdline, int argc, char **argv, std::vector<std::string> ini_config, resource::partitioner_mode rpmode = resource::mode_default, runtime_mode mode = runtime_mode_default)

partitioner (std::nullptr_t f, int argc, char **argv, resource::partitioner_mode rpmode = resource::mode_default, hpx::runtime_mode mode = hpx::runtime_mode_default)

partitioner (std::nullptr_t f, int argc, char **argv, std::vector<std::string> const &cfg, resource::partitioner_mode rpmode = resource::mode_default, hpx::runtime_mode mode = hpx::runtime_mode_default)

partitioner (std::nullptr_t f, hpx::program_options::options_description const &desc_cmdline, int argc, char **argv, std::vector<std::string> ini_config, resource::partitioner_mode rpmode = resource::mode_default, runtime_mode mode = runtime_mode_default)

void create_thread_pool (std::string const &name, scheduling_policy sched
= scheduling_policy::unspecified,
hpx::threads::policies::scheduler_mode
= hpx::threads::policies::scheduler_mode::default_mode)

void create_thread_pool (std::string const &name, scheduler_function scheduler_creation)

void set_default_pool_name (std::string const &name)

const std::string &get_default_pool_name () const

void add_resource (hpx::resource::pu const &p, std::string const &pool_name,
std::size_t num_threads = 1)

void add_resource (hpx::resource::pu const &p, std::string const &pool_name, bool exclusive, std::size_t num_threads = 1)

void add_resource (std::vector<hpx::resource::pu> const &pv, std::string const &pool_name, bool exclusive = true)

void add_resource (hpx::resource::core const &c, std::string const &pool_name, bool exclusive = true)
void add_resource(std::vector<hpx::resource::core> &cv, std::string const &pool_name, bool exclusive = true)
void add_resource(hpx::resource::numa_domain const &nd, std::string const &pool_name, bool exclusive = true)
void add_resource(std::vector<hpx::resource::numa_domain> const &ndv, std::string const &pool_name, bool exclusive = true)

std::vector<numa_domain> const &numa_domains() const
std::size_t get_number_requested_threads()

hpx::threads::topology const &get_topology() const

Private Members
detail::partitioner &partitioner_

class pu

Public Functions

pu(std::size_t id = invalid_pu_id, core *core = nullptr, std::size_t thread_occupancy = 0)
std::size_t id() const

Private Functions

std::vector<pu> pus_sharing_core()
std::vector<pu> pus_sharing_numa_domain()

Private Members

std::size_t id_
core *core_
std::size_t thread_occupancy_
std::size_t thread_occupancy_count_

Private Static Attributes

const std::size_t invalid_pu_id = std::size_t(-1)

Friends

friend hpx::resource::core
friend hpx::resource::numa_domain

checkpoint

#include <compatibility/hpx/util/checkpoint.hpp>

This header defines the save_checkpoint and restore_checkpoint functions. These functions are designed to help HPX application developer’s checkpoint their applications. Save_checkpoint serializes one or more objects and saves them as a byte stream. Restore_checkpoint converts the byte stream back into instances of the objects.
#include <hpx/checkpoint/checkpoint.hpp>

namespace hpx

namespace util

## Functions

\[ \text{std::ostream } \& \text{operator<<} \left( \text{std::ostream } \& \text{ost}, \text{checkpoint } \text{const } \& \text{ckp} \right) \]

Operator<< Overload

This overload is the main way to write data from a checkpoint to an object such as a file. Inside the function, the size of the checkpoint will be written to the stream before the checkpoint’s data. The operator>> overload uses this to read the correct number of bytes. Be mindful of this additional write and read when you use different facilities to write out or read in data to a checkpoint!

**Parameters**

- \( \text{ost} \): Output stream to write to.
- \( \text{ckp} \): Checkpoint to copy from.

**Return** Operator<< returns the ostream object.

\[ \text{std::istream } \& \text{operator>>} \left( \text{std::istream } \& \text{ist}, \text{checkpoint } \& \text{ckp} \right) \]

Operator>> Overload

This overload is the main way to read in data from an object such as a file to a checkpoint. It is important to note that inside the function, the first variable to be read is the size of the checkpoint. This size variable is written to the stream before the checkpoint’s data in the operator<< overload. Be mindful of this additional read and write when you use different facilities to read in or write out data from a checkpoint!

**Parameters**

- \( \text{ist} \): Input stream to write from.
- \( \text{ckp} \): Checkpoint to write to.

**Return** Operator>> returns the ostream object.

\[ \text{template } \langle \text{typename } T, \text{typename... } Ts, \text{typename } U = \text{typename std::enable_if!hpx::traits::is_launch_policy<T>::value && !std::is_same<typename std::decay<T>::type, checkpoint>::value>::type} \rangle \]

\text{hpx::future<checkpoint> save_checkpoint \left( T \&& t, Ts&&... \& ts \right)}

Save_checkpoint

Save_checkpoint takes any number of objects which a user may wish to store and returns a future to a checkpoint object. This function can also store a component either by passing a shared_ptr to the component or by passing a component’s client instance to save_checkpoint. Additionally the function can take a policy as a first object which changes its behavior depending on the policy passed to it. Most notably, if a sync policy is used save_checkpoint will simply return a checkpoint object.

**Template Parameters**

- \( T \): Containers passed to save_checkpoint to be serialized and placed into a checkpoint object.
- \( Ts \): More containers passed to save_checkpoint to be serialized and placed into a checkpoint object.
- \( U \): This parameter is used to make sure that \( T \) is not a launch policy or a checkpoint. This forces the compiler to choose the correct overload.

**Parameters**

- \( t \): A container to restore.
- \( ts \): Other containers to restore Containers must be in the same order that they were inserted into the checkpoint.

**Return** Save_checkpoint returns a future to a checkpoint with one exception: if you pass
hpx::launch::sync as the first argument. In this case save_checkpoint will simply return a checkpoint.

```cpp
template <typename T, typename... Ts>
hpx::future<checkpoint> save_checkpoint (checkpoint &&c, T &&t, Ts&&... ts)
```

**Save_checkpoint** - Take a pre-initialized checkpoint

Save_checkpoint takes any number of objects which a user may wish to store and returns a future to a checkpoint object. This function can also store a component either by passing a shared_ptr to the component or by passing a component’s client instance to save_checkpoint. Additionally the function can take a policy as a first object which changes its behavior depending on the policy passed to it. Most notably, if a sync policy is used save_checkpoint will simply return a checkpoint object.

**Template Parameters**

- **T**: Containers passed to save_checkpoint to be serialized and placed into a checkpoint object.
- **Ts**: More containers passed to save_checkpoint to be serialized and placed into a checkpoint object.

**Parameters**

- **c**: Takes a pre-initialized checkpoint to copy data into.
- **t**: A container to restore.
- **ts**: Other containers to restore Containers must be in the same order that they were inserted into the checkpoint.

**Return** Save_checkpoint returns a future to a checkpoint with one exception: if you pass hpx::launch::sync as the first argument. In this case save_checkpoint will simply return a checkpoint.

```cpp
template <typename T, typename... Ts>
hpx::future<checkpoint> save_checkpoint (hpx::launch p, T &&t, Ts&&... ts)
```

**Save_checkpoint** - Policy overload

Save_checkpoint takes any number of objects which a user may wish to store and returns a future to a checkpoint object. This function can also store a component either by passing a shared_ptr to the component or by passing a component’s client instance to save_checkpoint. Additionally the function can take a policy as a first object which changes its behavior depending on the policy passed to it. Most notably, if a sync policy is used save_checkpoint will simply return a checkpoint object.

**Template Parameters**

- **T**: Containers passed to save_checkpoint to be serialized and placed into a checkpoint object.
- **Ts**: More containers passed to save_checkpoint to be serialized and placed into a checkpoint object.

**Parameters**

- **p**: Takes an HPX launch policy. Allows the user to change the way the function is launched i.e. async, sync, etc.
- **t**: A container to restore.
- **ts**: Other containers to restore Containers must be in the same order that they were inserted into the checkpoint.

**Return** Save_checkpoint returns a future to a checkpoint with one exception: if you pass hpx::launch::sync as the first argument. In this case save_checkpoint will simply return a checkpoint.

```cpp
template <typename T, typename... Ts>
hpx::future<checkpoint> save_checkpoint (hpx::launch p, checkpoint &&c, T &&t, Ts&&... ts)
```

**Save_checkpoint** - Policy overload & pre-initialized checkpoint

Save_checkpoint takes any number of objects which a user may wish to store and returns a future to a checkpoint object. This function can also store a component either by passing a shared_ptr to the component or by passing a component’s client instance to save_checkpoint. Additionally the function
can take a policy as a first object which changes its behavior depending on the policy passed to it. Most notably, if a sync policy is used saveCheckpoint will simply return a checkpoint object.

**Template Parameters**
- `T`: Containers passed to `saveCheckpoint` to be serialized and placed into a checkpoint object.
- `Ts`: More containers passed to `saveCheckpoint` to be serialized and placed into a checkpoint object.

**Parameters**
- `p`: Takes an HPX launch policy. Allows the user to change the way the function is launched i.e. async, sync, etc.
- `c`: Takes a pre-initialized checkpoint to copy data into.
- `t`: A container to restore.
- `ts`: Other containers to restore Containers must be in the same order that they were inserted into the checkpoint.

**Return**
`saveCheckpoint` returns a future to a checkpoint with one exception: if you pass `hpx::launch::sync` as the first argument. In this case `saveCheckpoint` will simply return a checkpoint.

```cpp
template <typename T, typename... Ts, typename U = typename std::enable_if<!std::is_same< typename std::decay<T>::type, checkpoint>::value>::type>
checkpoint save_checkpoint (hpx::launch::sync_policy sync_p, T &&t, Ts&&... ts)
```

**SaveCheckpoint - Sync_policy overload**

`saveCheckpoint` takes any number of objects which a user may wish to store and returns a future to a checkpoint object. This function can also store a component either by passing a shared_ptr to the component or by passing a component’s client instance to `saveCheckpoint`. Additionally the function can take a policy as a first object which changes its behavior depending on the policy passed to it. Most notably, if a sync policy is used `saveCheckpoint` will simply return a checkpoint object.

**Template Parameters**
- `T`: Containers passed to `saveCheckpoint` to be serialized and placed into a checkpoint object.
- `Ts`: More containers passed to `saveCheckpoint` to be serialized and placed into a checkpoint object.
- `U`: This parameter is used to make sure that `T` is not a checkpoint. This forces the compiler to choose the correct overload.

**Parameters**
- `sync_p`: `hpx::launch::sync_policy`
- `t`: A container to restore.
- `ts`: Other containers to restore Containers must be in the same order that they were inserted into the checkpoint.

**Return**
`saveCheckpoint` which is passed `hpx::launch::sync_policy` will return a checkpoint which contains the serialized values checkpoint.

```cpp
template <typename T, typename... Ts>
checkpoint save_checkpoint (hpx::launch::sync_policy sync_p, checkpoint &&c, T &&t,
                           Ts&&... ts)
```

**SaveCheckpoint - Sync_policy overload & pre-init. checkpoint**

`saveCheckpoint` takes any number of objects which a user may wish to store and returns a future to a checkpoint object. This function can also store a component either by passing a shared_ptr to the component or by passing a component’s client instance to `saveCheckpoint`. Additionally the function can take a policy as a first object which changes its behavior depending on the policy passed to it. Most notably, if a sync policy is used `saveCheckpoint` will simply return a checkpoint object.

**Template Parameters**
- `T`: Containers passed to `saveCheckpoint` to be serialized and placed into a checkpoint object.
- `Ts`: More containers passed to `saveCheckpoint` to be serialized and placed into a checkpoint object.

**Parameters**
• **sync_p**: `hpx::launch::sync_policy`
• **c**: Takes a pre-initialized checkpoint to copy data into.
• **t**: A container to restore.
• **ts**: Other containers to restore. Containers must be in the same order that they were inserted into the checkpoint.

**Return** Save_checkpoint which is passed `hpx::launch::sync_policy` will return a checkpoint which contains the serialized values checkpoint.

```cpp
template <typename T, typename... Ts>
void restore_checkpoint (checkpoint const &c, T &t, Ts &... ts)
```

Restore_checkpoint takes a checkpoint object as a first argument and the containers which will be filled from the byte stream (in the same order as they were placed in `save_checkpoint`). Restore_checkpoint can resurrect a stored component in two ways: by passing in an instance of a component’s shared_ptr or by passing in an instance of the component’s client.

**Return** Restore_checkpoint returns void.

**Template Parameters**
• **T**: A container to restore.
• **Ts**: Other containers to restore. Containers must be in the same order that they were inserted into the checkpoint.

**Parameters**
• **c**: The checkpoint to restore.
• **t**: A container to restore.
• **ts**: Other containers to restore. Containers must be in the same order that they were inserted into the checkpoint.

### class checkpoint

```cpp
#include <checkpoint.hpp> Checkpoint Object
```

Checkpoint is the container object which is produced by `save_checkpoint` and is consumed by a `restore_checkpoint`. A checkpoint may be moved into the `save_checkpoint` object to write the byte stream to the pre-created checkpoint object.

Checkpoints are able to store all containers which are able to be serialized including components.

**Public Types**

```cpp
using const_iterator = std::vector::const_iterator
```

**Public Functions**

```cpp
checkpoint ()
cHECKPOINT (checkpoint const &c)
cHECKPOINT (checkpoint &c)
~checkPOint ()
cHECKPOINT (std::vector<char> const &vec)
cHECKPOINT (std::vector<char> &&vec)
CHECKPOINT &operator= (checkpoint const &c)
CHECKPOINT &operator= (checkpoint &c)
```
const_iterator begin() const
const_iterator end() const
size_t size() const

Private Functions

template <typename Archive>
void serialize(Archive &arch, const unsigned int version)

Private Members

std::vector<char> data_

Friends

friend hpx::util::hpx::serialization::access

std::ostream &operator<<(std::ostream &ost, checkpoint const &ckp)
Operator<< Overload

This overload is the main way to write data from a checkpoint to an object such as a file. Inside the function, the size of the checkpoint will be written to the stream before the checkpoint’s data. The operator>> overload uses this to read the correct number of bytes. Be mindful of this additional write and read when you use different facilities to write out or read in data to a checkpoint!

Parameters
• ost: Output stream to write to.
• ckp: Checkpoint to copy from.

Return Operator<< returns the ostream object.

std::istream &operator>>(std::istream &ist, checkpoint &ckp)
Operator>> Overload

This overload is the main way to read in data from an object such as a file to a checkpoint. It is important to note that inside the function, the first variable to be read is the size of the checkpoint. This size variable is written to the stream before the checkpoint’s data in the operator<< overload. Be mindful of this additional read and write when you use different facilities to read in or write out data from a checkpoint!

Parameters
• ist: Input stream to write from.
• ckp: Checkpoint to write to.

Return Operator>> returns the ostream object.

template <typename T, typename... Ts>
void restore_checkpoint(checkpoint const &ck, T &t, Ts&... ts)
Restore_checkpoint

Restore_checkpoint takes a checkpoint object as a first argument and the containers which will be filled from the byte stream (in the same order as they were placed in save_checkpoint). Restore_checkpoint can resurrect a stored component in two ways: by passing in an instance of a component’s shared_ptr or by passing in an instance of the component’s client.

Return Restore_checkpoint returns void.

Template Parameters
• T: A container to restore.
• **Ts**: Other containers to restore. Containers must be in the same order that they were inserted into the checkpoint.

**Parameters**

- **c**: The checkpoint to restore.
- **t**: A container to restore.
- **ts**: Other containers to restore. Containers must be in the same order that they were inserted into the checkpoint.

```cpp
bool operator==(checkpoint const &lhs, checkpoint const &rhs)
bool operator!=(checkpoint const &lhs, checkpoint const &rhs)
```

**affinity**

```cpp
#include <compatibility/hpx/runtime/threads/policies/parse_affinity_options.hpp>
#include <compatibility/hpx/runtime/threads/policies/affinity_data.hpp>
#include <hpx/affinity/parse_affinity_options.hpp>
namespace hpx{

namespace threads{

Functions

void parse_affinity_options(std::string const &spec, std::vector<mask_type> &affinities, std::size_t used_cores, std::size_t max_cores, std::size_t num_threads, std::vector<std::size_t> &num_pus, bool use_process_mask, error_code &ec = throws)
void parse_affinity_options(std::string const &spec, std::vector<mask_type> &affinities, error_code &ec = throws)

#include <hpx/affinity/affinity_data.hpp>

memory

#include <hpx/memory/intrusive_ptr.hpp>

template <typename T>
template<>
struct hash<hpx::memory::intrusive_ptr<T>>

Public Types

template<>
using result_type = std::size_t

Public Functions

result_type operator() (hpx::memory::intrusive_ptr<T> const &p) const

namespace hpx{
namespace memory

Functions

```cpp
template <typename T, typename U>
bool operator== (intrusive_ptr<T> const &a, intrusive_ptr<U> const &b)

template <typename T, typename U>
bool operator!= (intrusive_ptr<T> const &a, intrusive_ptr<U> const &b)

template <typename T, typename U>
bool operator== (intrusive_ptr<T> const &a, U *b)

template <typename T, typename U>
bool operator!= (intrusive_ptr<T> const &a, U *b)

template <typename T, typename U>
bool operator== (T *a, intrusive_ptr<U> const &b)

template <typename T, typename U>
bool operator!= (T *a, intrusive_ptr<U> const &b)

template <typename T>
bool operator== (intrusive_ptr<T> const &p, std::nullptr_t)

template <typename T>
bool operator== (std::nullptr_t, intrusive_ptr<T> const &p)

template <typename T>
bool operator!= (intrusive_ptr<T> const &p, std::nullptr_t)

template <typename T>
bool operator!= (std::nullptr_t, intrusive_ptr<T> const &p)

template <typename T>
bool operator< (intrusive_ptr<T> const &a, intrusive_ptr<T> const &b)

template <typename T>
void swap (intrusive_ptr<T> &lhs, intrusive_ptr<T> &rhs)

template <typename T>
T *get_pointer (intrusive_ptr<T> const &p)

template <typename T, typename U>
intrusive_ptr<T> static_pointer_cast (intrusive_ptr<U> const &p)

template <typename T, typename U>
intrusive_ptr<T> const_pointer_cast (intrusive_ptr<U> const &p)

template <typename T, typename U>
intrusive_ptr<T> dynamic_pointer_cast (intrusive_ptr<U> const &p)

template <typename T, typename U>
intrusive_ptr<T> static_pointer_cast (intrusive_ptr<U> &&p)

template <typename T, typename U>
intrusive_ptr<T> const_pointer_cast (intrusive_ptr<U> &&p)

template <typename T, typename U>
intrusive_ptr<T> dynamic_pointer_cast (intrusive_ptr<U> &&p)

template <typename Y>
std::ostream &operator<< (std::ostream &os, intrusive_ptr<Y> const &p)
```
template <typename T>
class intrusive_ptr

Public Types

template<>
using element_type = T

Public Functions

intrusive_ptr()
intrusive_ptr(T *p, bool add_ref = true)

intrusive_ptr<U, typename Enable = typename std::enable_if< memory::detail::sp_convertible<U, T>::value>::type>::intrusive_ptr(intrusive_ptr<U> const &rhs)

intrusive_ptr(intrusive_ptr const &rhs)

~intrusive_ptr()

template <typename U>
intrusive_ptr &operator=(intrusive_ptr<U> const &rhs)

intrusive_ptr &operator=(intrusive_ptr &rhs)

template <typename U, typename Enable = typename std::enable_if< memory::detail::sp_convertible<U, T>::value>::type>
intrusive_ptr(U) &&
intrusive_ptr<U> &&

intrusive_ptr &operator=(intrusive_ptr &&rhs)
intrusive_ptr &operator=(T *rhs)
void reset()
void reset(T *rhs)
void reset(T *rhs, bool add_ref)

T *get() const
T *detach()
T &operator*() const
T *operator->() const
operator bool() const
void swap(intrusive_ptr &rhs)

Private Types

template<>
using this_type = intrusive_ptr
Private Members

T *px

Friends

friend hpx::memory::intrusive_ptr

namespace std

    template<typename T>
    template<>
    struct hash<hpx::memory::intrusive_ptr<T>>

Public Types

    template<>
    using result_type = std::size_t

Public Functions

    result_type operator() (hpx::memory::intrusive_ptr<T> const &p) const

#include <hpx/memory/serialization/intrusive_ptr.hpp>

namespace hpx

namespace serialization

Functions

    template<typename T>
    void load (input_archive &ar, hpx::intrusive_ptr<T> &ptr, unsigned)

    template<typename T>
    void save (output_archive &ar, hpx::intrusive_ptr<T> const &ptr, unsigned)

hpx::serialization::HPX_SERIALIZATION_SPLIT_FREE_TEMPLATE((template< typename T >), ( hpx::intrusive_ptr < T >))

#include <hpx/program_options.hpp>

#include <hpx/program_options/parsers.hpp>

namespace hpx

namespace program_options

Typedefs

    using hpx::program_options::parsed_options = typedef basic_parsed_options<char>

    using hpx::program_options::wparsed_options = typedef basic_parsed_options<wchar_t>
using hpx::program_options::ext_parser = typedef std::function<std::pair<std::string, std::string>(const std::string&)> Augments basic_parsed_options<wchar_t> with conversion from 'parsed_options' 

using hpx::program_options::command_line_parser = typedef basic_command_line_parser<char>
using hpx::program_options::wcommand_line_parser = typedef basic_command_line_parser<wchar_t>

Enums

enum collect_unrecognized_mode 
Controls if the ‘collect_unregistered’ function should include positional options, or not.
Values:
include_positional
exclude_positional

Functions

template <class Char>
basic_parsed_options<Char> parse_command_line (int argc, const Char *const argv[],
const options_description&, int style = 0, std::function<std::pair<std::string, std::string>(std::string&)> const std::string& > ext = ext_parser() Creates instance of ‘command_line_parser’, passes parameters to it, and returns the result of calling the ‘run’ method.

template <class Char>
basic_parsed_options<Char> parse_config_file (std::basic_istream<Char>&,
const options_description&, bool allow_unregistered = false)
Parse a config file.
Read from given stream.

template <class Char = char>
basic_parsed_options<Char> parse_config_file (const char *filename, const options_description&, bool allow_unregistered = false)
Parse a config file.
Read from file with the given name. The character type is passed to the file stream.

template <class Char>
std::vector<std::basic_string<Char>> collect_unrecognized (const std::vector<basic_option<Char>>& options, enum collect_unrecognized_mode mode)
Collects the original tokens for all named options with ‘unregistered’ flag set. If ‘mode’ is ‘include_positional’ also collects all positional options. Returns the vector of original tokens for all collected options.

parsed_options parse_environment (const options_description&, const std::function<std::string>(std::string) std::string > &name_mapper Parse environment.
For each environment variable, the ‘name_mapper’ function is called to obtain the option name. If it returns empty string, the variable is ignored.
This is done since naming of environment variables is typically different from the naming of command line options.

```cpp
parsed_options parse_environment (const options_description&, const std::string &prefix)
```

Parse environment.

Takes all environment variables which start with ‘prefix’. The option name is obtained from variable name by removing the prefix and converting the remaining string into lower case.

```cpp
parsed_options parse_environment (const options_description&, const char *prefix)
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts. This function exists to resolve ambiguity between the two above functions when second argument is of ‘char*’ type. There’s implicit conversion to both std::function and string.

```cpp
std::vector<std::string> split_unix (const std::string &cmdline)
```

Splits a given string to a collection of single strings which can be passed to command_line_parser. The second parameter is used to specify a collection of possible separator chars used for splitting. The separator is defaulted to space “ “. Splitting is done in a unix style way, with respect to quotes “” and escape characters ‘\’.

```cpp
std::vector<std::wstring> split_unix (const std::wstring &cmdline)
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

```cpp
template <class Char>
class basic_command_line_parser : cmdline
#include <parsers.hpp>
```

Command line parser.

The class allows one to specify all the information needed for parsing and to parse the command line. It is primarily needed to emulate named function parameters – a regular function with 5 parameters will be hard to use and creating overloads with a smaller number of parameters will be confusing.

For the most common case, the function parse_command_line is a better alternative.

There are two typedefs – command_line_parser and wcommand_line_parser, for charT == char and charT == wchar_t cases.

### Public Functions

```cpp
basic_command_line_parser (const std::vector<std::basic_string<Char>> &args)
```

Creates a command line parser for the specified arguments list. The ‘args’ parameter should not include program name.

```cpp
basic_command_line_parser (int argc, const Char *const argv[])
```

Creates a command line parser for the specified arguments list. The parameters should be the same as passed to ‘main’.

```cpp
basic_command_line_parser &options (const options_description &desc)
```

Sets options descriptions to use.

```cpp
basic_command_line_parser &positional (const positional_options_description &desc)
```

Sets positional options description to use.

```cpp
basic_command_line_parser &style (int)
```

Sets the command line style.
basic_command_line_parser &extra_parser_(ext_parser)
Sets the extra parsers.

basic_parsed_options<Char> run()
Parses the options and returns the result of parsing. Throws on error.

basic_command_line_parser &allow_unregistered()
Specifies that unregistered options are allowed and should be passed through. For each command
like token that looks like an option but does not contain a recognized name, an instance of ba-
sic_option<Char> will be added to result, with 'unrecognized' field set to 'true'. It’s possible to
collect all unrecognized options with the 'collect_unrecognized' function.

Private Members

case options_description *m_desc

template <class Char>
class basic_parsed_options
#include <parsers.hpp> Results of parsing an input source. The primary use of this class is passing
information from parsers component to value storage component. This class does not makes much
sense itself.

Public Functions

basic_parsed_options (const options_description *xdescription, int options_prefix = 0)

Public Members

std::vector<basic_option<Char>> options
Options found in the source.

const options_description *description
Options description that was used for parsing. Parsers should return pointer to the instance of
option_description passed to them, and issues of lifetime are up to the caller. Can be NULL.

int m_options_prefix
Mainly used for the diagnostic messages in exceptions. The canonical option prefix for the parser
which generated these results, depending on the settings for basic_command_line_parser::style() or
cmdline::style(). In order of precedence of command_line_style enums: allow_long allow_long_disguise allow_dash_for_short allow_slash_for_short

template <>
template<>
class basic_parsed_options<wchar_t>
#include <parsers.hpp> Specialization of basic_parsed_options which:
• provides convenient conversion from basic_parsed_options<Char>
• stores the passed char-based options for later use.

Public Functions

basic_parsed_options (const basic_parsed_options<Char> &po)
Constructs wrapped options from options in UTF8 encoding.
Public Members

```
std::vector<basic_option<wchar_t>> options
const options_description * description
basic_parsed_options<char> utf8_encoded_options
Stores UTF8 encoded options that were passed to constructor, to avoid reverse conversion in some cases.
int m_options_prefix
Mainly used for the diagnostic messages in exceptions. The canonical option prefix for the parser which generated these results, depending on the settings for basic_command_line_parser::style() or cmdline::style(). In order of precedence of command_line_style enums: allow_long allow_long_disguise allow_dash_for_short allow_slash_for_short
```

```c
#include <hpx/program_options/variables_map.hpp>

namespace hpx

namespace program_options

Functions

void store(const basic_parsed_options<char> &options, variables_map &m, bool utf8 = false)
Stores in ‘m’ all options that are defined in ‘options’. If ‘m’ already has a non-defaulted value of an option, that value is not changed, even if ‘options’ specify some value.

void store(const basic_parsed_options<wchar_t> &options, variables_map &m)
Stores in ‘m’ all options that are defined in ‘options’. If ‘m’ already has a non-defaulted value of an option, that value is not changed, even if ‘options’ specify some value. This is wide character variant.

void notify(variables_map &m)
Runs all ‘notify’ function for options in ‘m’.

class abstract_variables_map
#include <variables_map.hpp> Implements string->string mapping with convenient value casting facilities.
Subclassed by hpx::program_options::variables_map

Public Functions

abstract_variables_map()

abstract_variables_map(const abstract_variables_map *next)

virtual ~abstract_variables_map()

const variable_value &operator[] (const std::string &name) const
Obtains the value of variable ‘name’, from *this and possibly from the chain of variable maps.
- if there’s no value in *this.
- if there’s next variable map, returns value from it
- otherwise, returns empty value
- if there’s defaulted value
- if there’s next variable map, which has a non-defaulted value, return that
– otherwise, return value from *this
• if there’s a non-defaulted value, returns it.

void **next**(abstract_variables_map *next)

Sets next variable map, which will be used to find variables not found in *this.

**Private Functions**

**virtual const variable_value &get (const std::string &name) const = 0**

Returns value of variable ‘name’ stored in *this, or empty value otherwise.

**Private Members**

**const abstract_variables_map *m_next**

**class variable_value**

#include <variables_map.hpp> Class holding value of option. Contains details about how the value is set and allows to conveniently obtain the value.

**Public Functions**

**variable_value ()**

**variable_value (const hpx::util::any_nonser &xv, bool xdefaulted)**

**template <class T>**

**const T &as () const**

If stored value if of type T, returns that value. Otherwise, throws boost::bad_any_cast exception.

**template <class T>**

**T &as ()**

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

**bool empty () const**

Returns true if no value is stored.

**bool defaulted () const**

Returns true if the value was not explicitly given, but has default value.

**const hpx::util::any_nonser &value () const**

Returns the contained value.

**hpx::util::any_nonser &value ()**

Returns the contained value.

**Private Members**

**hpx::util::any_nonser v**

**bool m_defaulted**

**std::shared_ptr<const value_semantic> m_value_semantic**
**Friends**

```cpp
friend hpx::program_options::variables_map
void store (const basic_parsed_options<char> &options, variables_map &m, bool utf8)
    Stores in ‘m’ all options that are defined in ‘options’. If ‘m’ already has a non-defaulted value of
    an option, that value is not changed, even if ‘options’ specify some value.
```

**class variables_map**

```cpp
#include <variables_map.hpp>
Concrete variables map which store variables in real map.
This class is derived from std::map<std::string, variable_value>, so you can use all map operators to
examine its content.
```

**Public Functions**

```cpp
variables_map()
variables_map(const abstract_variables_map *next)
const variable_value &operator[](const std::string &name) const
void clear()
void notify()
```

**Private Functions**

```cpp
const variable_value &get(const std::string &name) const
    Implementation of abstract_variables_map::get which does ‘find’ in *this.
```

**Private Members**

```cpp
std::set<std::string> m_final
    Names of option with ‘final’ values – which should not be changed by subsequence assignments.
std::map<std::string, std::string> m_required
    Names of required options, filled by parser which has access to options_description. The map values
    are the “canonical” names for each corresponding option. This is useful in creating diagnostic
    messages when the option is absent.
```

**Friends**

```cpp
#include <hpx/program_options/options_description.hpp>
namespace hpx
namespace program_options
```

```cpp
class duplicate_option_error: public hpx::program_options::error
    #include <options_description.hpp> Class thrown when duplicate option description is found.
```
Public Functions

duplicate_option_error (const std::string &xwhat)

class option_description
#include <options_description.hpp> Describes one possible command line/config file option. There are two kinds of properties of an option. First describe it syntactically and are used only to validate input. Second affect interpretation of the option, for example default value for it or function that should be called when the value is finally known. Routines which perform parsing never use second kind of properties – they are side effect free.
See options_description

Public Types

enum match_result
Values:
  no_match
  full_match
  approximate_match

Public Functions

option_description()

option_description (const char *name, const value_semantic *s)
  Initializes the object with the passed data.
  Note: it would be nice to make the second parameter auto_ptr, to explicitly pass ownership. Unfortunately, it’s often needed to create objects of types derived from ‘value_semantic’:
    options_description d; d.add_options()("a", parameter<int>(“n”)->default_value(1)); Here, the static type returned by ‘parameter’ should be derived from value_semantic.

Alas, derived->base conversion for auto_ptr does not really work, see http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2000/n1232.pdf http://www.open-std.org/jtc1/sc22/wg21/docs/cwg_defects.html#84

So, we have to use plain old pointers. Besides, users are not expected to use the constructor directly.

The ‘name’ parameter is interpreted by the following rules:
• if there’s no “,” character in ‘name’, it specifies long name
• otherwise, the part before “,” specifies long name and the part after – short name.

option_description (const char *name, const value_semantic *s, const char *description)
  Initializes the class with the passed data.

virtual ~option_description ()

match_result match (const std::string &option, bool approx, bool long_ignore_case, bool short_ignore_case) const
  Given ‘option’, specified in the input source, returns ‘true’ if ‘option’ specifies *this.

const std::string &key (const std::string &option) const
  Returns the key that should identify the option, in particular in the variables_map class. The ‘option’ parameter is the option spelling from the input source. If option name contains ‘*‘,
returns ‘option’. If long name was specified, it’s the long name, otherwise it’s a short name with
pre-pended ‘-’.

`std::string canonical_display_name (int canonical_option_style = 0) const`
Returns the canonical name for the option description to enable the user to recognized a matching
option. 1) For short options (‘-’, ‘/’), returns the short name prefixed. 2) For long options (‘’ / ‘’)
returns the first long name prefixed 3) All other cases, returns the first long name (if present) or
the short name, un-prefixed.

`const std::string &long_name () const`

`const std::pair<const std::string *, std::size_t> long_names () const`

`const std::string &description () const`
Explanation of this option.

`std::shared_ptr<const value_semantic> semantic () const`
Semantic of option’s value.

`std::string format_name () const`
Returns the option name, formatted suitably for usage message.

`std::string format_parameter () const`
Returns the parameter name and properties, formatted suitably for usage message.

Private Functions

`option_description &set_names (const char *name)`

Private Members

`std::string m_short_name`
a one-character “switch” name - with its prefix, so that this is either empty or has length 2 (e.g.
‘’

`std::vector<std::string> m_long_names`
one or more names by which this option may be specified on a command-line or in a config file,
which are not a single-letter switch. The names here are without any prefix.

`std::string m_description`

`std::shared_ptr<const value_semantic> m_value_semantic`

class options_description
#include <options_description.hpp> A set of option descriptions. This provides convenient interface
for adding new option (the add_options) method, and facilities to search for options by name.

See here for option adding interface discussion.
See option_description

Public Functions

`options_description (unsigned line_length = m_default_line_length, unsigned
  min_description_length = m_default_line_length / 2)`
Creates the instance.
options_description(const std::string &caption, unsigned line_length = m_default_line_length, unsigned min_description_length = m_default_line_length / 2)

Creates the instance. The ‘caption’ parameter gives the name of this ‘options_description’ instance. Primarily useful for output. The ‘description_length’ specifies the number of columns that should be reserved for the description text; if the option text encroaches into this, then the description will start on the next line.

void add(std::shared_ptr<option_description> desc)

Adds new variable description. Throws duplicate_variable_error if either short or long name matches that of already present one.

options_description &add(const options_description &desc)

Adds a group of option description. This has the same effect as adding all option_descriptions in ‘desc’ individually, except that output operator will show a separate group. Returns *this.

std::size_t get_option_column_width() const

Find the maximum width of the option column, including options in groups.

options_description_easy_init add_options()

Returns an object of implementation-defined type suitable for adding options to options_description. The returned object will have overloaded operator() with parameter type matching ‘option_description’ constructors. Calling the operator will create new option_description instance and add it.

const option_description &find(const std::string &name, bool approx, bool long_ignore_case = false, bool short_ignore_case = false) const

const option_description *find_nothrow(const std::string &name, bool approx, bool long_ignore_case = false, bool short_ignore_case = false) const

const std::vector<std::shared_ptr<option_description>> &options() const

void print (std::ostream &os, std::size_t width = 0) const

Outputs ‘desc’ to the specified stream, calling ‘f’ to output each option_description element.

Public Static Attributes

const unsigned m_default_line_length

Private Types

using name2index_iterator = std::map<std::string, int>::const_iterator

using approximation_range = std::pair<name2index_iterator, name2index_iterator>

Private Members

std::string m_caption

std::size_t const m_line_length

std::size_t const m_min_description_length

std::vector<std::shared_ptr<option_description>> m_options

std::vector<char> belong_to_group
std::vector<std::shared_ptr<options_description>> groups

Friends

std::ostream &operator<<(std::ostream &os, const options_description &desc)
   Produces a human readable output of ‘desc’, listing options, their descriptions and allowed parameters. Other options_description instances previously passed to add will be output separately.

class options_description_easy_init
#include <options_description.hpp> Class which provides convenient creation syntax to options_description.

Public Functions

options_description_easy_init(options_description *owner)

options_description_easy_init &operator()(const char *name, const char *description)

options_description_easy_init &operator()(const char *name, const value_semantic *s)

options_description_easy_init &operator()(const char *name, const value_semantic *s, const char *description)

Private Members

options_description *owner

#include <hpx/program_options/positional_options.hpp>

namespace hpx

namespace program_options

class positional_options_description
#include <positional_options.hpp> Describes positional options.

The class allows to guess option names for positional options, which are specified on the command line and are identified by the position. The class uses the information provided by the user to associate a name with every positional option, or tell that no name is known.

The primary assumption is that only the relative order of the positional options themselves matters, and that any interleaving ordinary options don’t affect interpretation of positional options.

The user initializes the class by specifying that first N positional options should be given the name X1, following M options should be given the name X2 and so on.

Public Functions

positional_options_description()

positional_options_description &add(const char *name, int max_count)
   Species that up to ‘max_count’ next positional options should be given the ‘name’. The value of ‘-1’ means ‘unlimited’. No calls to ‘add’ can be made after call with ‘max_value’ equal to ‘-1’.
unsigned `max_total_count()` const
Returns the maximum number of positional options that can be present. Can return (numeric_limits<unsigned>::max)() to indicate unlimited number.

`const std::string &name_for_position(unsigned position) const`
Returns the name that should be associated with positional options at ‘position’. Precondition: position < `max_total_count()`

**Private Members**

`std::vector<std::string> m_names`
`std::string m_trailing`

```cpp
#include <hpx/program_options/errors.hpp>
namespace hpx

namespace program_options

Functions
```

`std::string strip_prefixes(const std::string &text)`

```cpp
class ambiguous_option : public hpx::program_options::error_with_no_option_name
#include <errors.hpp> Class thrown when there’s ambiguity among several possible options.

Public Functions
```

`ambiguous_option(const std::vector<std::string> &alternatives)`
`~ambiguous_option()`
`const std::vector<std::string> &alternatives() const`

```cpp
Protected Functions
```

`void substitute_placeholders(const std::string &error_template) const`
Makes all substitutions using the template

```cpp
Private Members
```

`std::vector<std::string> m_alternatives`

```cpp
class error : public logic_error
#include <errors.hpp> Base class for all errors in the library.

Subclassed by hpx::program_options::duplicate_option_error, hpx::program_options::error_with_option_name, hpx::program_options::invalid_command_line_style, hpx::program_options::reading_file, hpx::program_options::too_many_positional_options_error
```
Public Functions

`error(const std::string &xwhat)`

class `error_with_no_option_name`: public `hpx::program_options::error_with_option_name`

#include `<errors.hpp>` Base class of un-parsable options, when the desired option cannot be identified.

It makes no sense to have an option name, when we can’t match an option to the parameter

Having this a part of the `error_with_option_name` hierarchy makes error handling a lot easier, even if the name indicates some sort of conceptual dissonance!

Subclassed by `hpx::program_options::ambiguous_option, hpx::program_options::unknown_option`

Public Functions

`error_with_no_option_name(const std::string &template_, const std::string &original_token = "")`

void `set_option_name(const std::string&)`  
  Does NOT set option name, because no option name makes sense

`~error_with_no_option_name()`

class `error_with_option_name`: public `hpx::program_options::error`

#include `<errors.hpp>` Base class for most exceptions in the library.

Substitutes the values for the parameter name placeholders in the template to create the human readable error message

Placeholders are surrounded by % signs: example% Poor man’s version of boost::format

If a parameter name is absent, perform default substitutions instead so ugly placeholders are never left in-place.

Options are displayed in “canonical” form This is the most unambiguous form of the parsed option name and would correspond to `option_description::format_name()` i.e. what is shown by `print_usage()`

The “canonical” form depends on whether the option is specified in short or long form, using dashes or slashes or without a prefix (from a configuration file)

Subclassed by `hpx::program_options::error_with_no_option_name, hpx::program_options::invalid_syntax, hpx::program_options::multiple_occurrences, hpx::program_options::multiple_values, hpx::program_options::required_option, hpx::program_options::validation_error`

Public Functions

`error_with_option_name(const std::string &template_, const std::string &option_name = ", const std::string &original_token = ", int option_style = 0)`

`~error_with_option_name()`
  gcc says that throw specification on dtor is loosened without this line

void `set_substitute(const std::string &parameter_name, const std::string &value)`  
  Substitute parameter_name->value to create the error message from the error template
void set_substitute_default (const std::string &parameter_name, const std::string &from, const std::string &to)

If the parameter is missing, then make the from->to substitution instead

void add_context (const std::string &option_name, const std::string &original_token, int option_style)

Add context to an exception

void set_prefix (int option_style)

virtual void set_option_name (const std::string &option_name)

Overridden in error_with_no_option_name

std::string get_option_name () const

void set_original_token (const std::string &original_token)

const char *what () const

Creates the error_message on the fly Currently a thin wrapper for substitute_placeholders()

Public Members

std::string m_error_template

template with placeholders

Protected Types

using string_pair = std::pair<std::string, std::string>

Protected Functions

virtual void substitute_placeholders (const std::string &error_template) const

Makes all substitutions using the template

void replace_token (const std::string &from, const std::string &to) const

std::string get_canonical_option_name () const

Construct option name in accordance with the appropriate prefix style: i.e. long dash or short slash etc

std::string get_canonical_option_prefix () const

Protected Attributes

int m_option_style

can be 0 = no prefix (config file options) allow_long allow_dash_for_short allow_slash_for_short allow_long_disguise

std::map<std::string, std::string> m_substitutions

substitutions from placeholders to values

std::map<std::string, string_pair> m_substitution_defaults

std::string m_message

Used to hold the error text returned by what()

class invalid_bool_value : public hpx::program_options::validation_error

#include <errors.hpp> Class thrown if there is an invalid bool value given
HPX Documentation, 1.4.1

Public Functions

invalid_bool_value (const std::string &value)

class invalid_command_line_style: public hpx::program_options::error #include <errors.hpp> Class thrown when there are programming error related to style

Public Functions

invalid_command_line_style (const std::string &msg)

class invalid_command_line_syntax: public hpx::program_options::invalid_syntax #include <errors.hpp> Class thrown when there are syntax errors in given command line

Public Functions

invalid_command_line_syntax (kind_t kind, const std::string &option_name = "", const std::string &original_token = "", int option_style = 0)

~invalid_command_line_syntax ()

class invalid_config_file_syntax: public hpx::program_options::invalid_syntax

Public Functions

invalid_config_file_syntax (const std::string &invalid_line, kind_t kind)

~invalid_config_file_syntax ()

std::string tokens () const

Convenience functions for backwards compatibility

class invalid_option_value: public hpx::program_options::validation_error #include <errors.hpp> Class thrown if there is an invalid option value given

Public Functions

invalid_option_value (const std::string &value)

invalid_option_value (const std::wstring &value)

class invalid_syntax: public hpx::program_options::error_with_option_name #include <errors.hpp> Class thrown when there’s syntax error either for command line or config file options. See derived children for concrete classes.

Subclassed by hpx::program_options::invalid_command_line_syntax, hpx::program_options::invalid_config_file_syntax

Public Types

enum kind_t

Values:

long_not_allowed = 30
long_adjacent_not_allowed

Chapter 2. What’s so special about HPX?
short_adjacent_not_allowed
empty_adjacent_parameter
missing_parameter
extra_parameter
unrecognized_line

Public Functions

invalid_syntax (kind_t kind, const std::string &option_name = "", const std::string &original_token = "", int option_style = 0)

~invalid_syntax()

kind_t kind() const

virtual std::string tokens() const
   Convenience functions for backwards compatibility

Protected Functions

std::string get_template (kind_t kind)
   Used to convert kind_t to a related error text

Protected Attributes

kind_t m_kind

class multiple_occurrences: public hpx::program_options::error_with_option_name
#include <errors.hpp> Class thrown when there are several occurrences of an option, but user called a method which cannot return them all.

Public Functions

multiple_occurrences()

~multiple_occurrences()

class multiple_values: public hpx::program_options::error_with_option_name
#include <errors.hpp> Class thrown when there are several option values, but user called a method which cannot return them all.

Public Functions

multiple_values()

~multiple_values()

class reading_file: public hpx::program_options::error
#include <errors.hpp> Class thrown if config file can not be read
Public Functions

reading_file (const char *filename)

class required_option: public hpx::program_options::error_with_option_name
#include <errors.hpp> Class thrown when a required/mandatory option is missing

Public Functions

required_option (const std::string &option_name)
~required_option ()

class too_many_positional_options_error: public hpx::program_options::error
#include <errors.hpp> Class thrown when there are too many positional options. This is a programming error.

Public Functions

too_many_positional_options_error ()

class unknown_option: public hpx::program_options::error_with_no_option_name
#include <errors.hpp> Class thrown when option name is not recognized.

Public Functions

unknown_option (const std::string &original_token = "")
~unknown_option ()

class validation_error: public hpx::program_options::error_with_option_name
#include <errors.hpp> Class thrown when value of option is incorrect.
Subclassed by hpx::program_options::invalid_bool_value, hpx::program_options::invalid_option_value

Public Types

enum kind_t
Values:
multiple_values_not_allowed = 30
at_least_one_value_required
invalid_bool_value
invalid_option_value
invalid_option

Public Functions

validation_error(kind_t kind, const std::string &option_name = ", const std::string &original_token = "", int option_style = 0)
~validation_error ()

kind_t kind () const
**Protected Functions**

```cpp
std::string get_template(kind_t kind)
```

Used to convert kind_t to a related error text

**Protected Attributes**

```cpp
kind_t m_kind
```

#include <hpx/program_options/force_linking.hpp>

namespace hpx

namespace program_options

**Typedefs**

```cpp
using hpx::program_options::parse_environment1_type = typedef basic_parsed_options<char> (*)( options_description const&, char const*)
using hpx::program_options::parse_environment2_type = typedef basic_parsed_options<char> (*)( options_description const&, std::function<std::string(std::string)> const&)
using hpx::program_options::parse_environment3_type = typedef basic_parsed_options<char> (*)( options_description const&, std::string const&)
using hpx::program_options::parse_config_file_char1_type = typedef basic_parsed_options<char> (*)( char const*, options_description const&, bool)
using hpx::program_options::parse_config_file_char2_type = typedef basic_parsed_options<char> (*)( std::basic_istream<char, struct std::char_traits<char> >&, options_description const&, bool)
using hpx::program_options::parse_config_file_wchar1_type = typedef basic_parsed_options<wchar_t> (*)( std::basic_istream<wchar_t, struct std::char_traits<wchar_t> >&, options_description const&, bool)
using hpx::program_options::split_unix_type = typedef std::vector<std::string> (*)(std::string const&, std::string const&, std::string const&, std::string const&)
```

**Functions**

```cpp
force_linking_helper &force_linking()
```

**Public Members**

```cpp
parse_environment1_type parse_environment1
parse_environment2_type parse_environment2
parse_environment3_type parse_environment3
parse_config_file_char1_type parse_config_file_char1
parse_config_file_char2_type parse_config_file_char2
parse_config_file_wchar1_type parse_config_file_wchar1
split_unix_type split_unix
```

#include <hpx/program_options/version.hpp>

**Defines**

```cpp
HPX_PROGRAM_OPTIONS_VERSION
```

The version of the source interface. The value will be incremented whenever a change is made which might cause compilation errors for existing code.
The `eof_iterator` class is useful for constructing forward iterators in cases where iterator extract data from some source and it's easy to detect `eof` – i.e. the situation where there's no data. One apparent example is reading lines from a file.

Implementing such iterators using `iterator_facade` directly would require to create class with three core operation, a couple of constructors. When using `eof_iterator`, the derived class should define only one method to get new value, plus a couple of constructors.

The basic idea is that iterator has `eof` bit. Two iterators are equal only if both have their `eof` bits set. The `get` method either obtains the new value or sets the `eof` bit.

Specifically, derived class should define:

1. A default constructor, which creates iterator with `eof` bit set. The constructor body should call `found_eof` method defined here.
2. Some other constructor. It should initialize some `data pointer` used in iterator operation and then call `get`.
3. The `get` method. It should operate this way:
   - look at some `data pointer` to see if new element is available; if not, it should call `found_eof`.
   - extract new element and store it at location returned by the `value` method.
   - advance the data pointer.

Essentially, the `get` method has the functionality of both `increment` and `dereference`. It's very good for the cases where data extraction implicitly moves data pointer, like for stream operation.

**Public Functions**

```
eof_iterator()
```

**Protected Functions**

```
ValueType &value()
```

Returns the reference which should be used by derived class to store the next value.
void `found_eof`()
    Should be called by derived class to indicate that it can’t produce next element.

**Private Functions**

void `increment`()

bool `equal` (const `eof_iterator` &other) const

const `ValueType` &`dereference`() const

**Private Members**

bool `m_at_eof`

`ValueType` `m_value`

**Friends**

friend `hpx::program_options::hpx::util::iterator_core_access`

#include `<hpx/program_options/environment_iterator.hpp>`

namespace `hpx`

namespace `program_options`

    class `environment_iterator` : public `hpx::program_options::eof_iterator<environment_iterator, std::pair<std::string, std::string>>`

**Public Functions**

`environment_iterator`(char **`environment`)

`environment_iterator()`

void `get`()

**Private Members**

char **`m_environment`

#include `<hpx/program_options/value_semantic.hpp>`

namespace `hpx`

namespace `program_options`

    Functions

    template <class `T`>
Creates a typed_value<T> instance. This function is the primary method to create value_semantic instance for a specific type, which can later be passed to 'option_description' constructor. The second overload is used when it’s additionally desired to store the value of option into program variable.

```cpp
template <class T>
typed_value<T> *value (T *v)
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

```cpp
template <class T>
typed_value<T, wchar_t> *wvalue ()
```

Creates a typed_value<T> instance. This function is the primary method to create value_semantic instance for a specific type, which can later be passed to 'option_description' constructor.

```cpp
template <class T>
typed_value<T, wchar_t> *wvalue (T *v)
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

```cpp
typed_value<bool> *bool_switch ()
```

Works the same way as the 'value<bool>' function, but the created value_semantic won’t accept any explicit value. So, if the option is present on the command line, the value will be 'true'.

```cpp
typed_value<bool> *bool_switch (bool *v)
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

```cpp
template <class T, class Char = char>
class typed_value : public hpx::program_options::value_semantic_codecvt_helper<Char>, public hpx::program_options::value_semantic_base
#include <value_semantic.hpp> Class which handles value of a specific type.
```

### Public Functions

```cpp
typed_value (T *store_to)
```

Ctor. The 'store_to' parameter tells where to store the value when it's known. The parameter can be NULL.

```cpp
typed_value *default_value (const T &v)
```

Specifies default value, which will be used if none is explicitly specified. The type ‘T’ should provide operator<< for ostream.

```cpp
typed_value *default_value (const T &v, const std::string &textual)
```

Specifies default value, which will be used if none is explicitly specified. Unlike the above overload, the type ‘T’ need not provide operator<< for ostream, but textual representation of default value must be provided by the user.

```cpp
typed_value *implicit_value (const T &v)
```

Specifies an implicit value, which will be used if the option is given, but without an adjacent value. Using this implies that an explicit value is optional.

```cpp
typed_value *value_name (const std::string &name)
```

Specifies the name used to to the value in help message.

```cpp
typed_value *implicit_value (const T &v, const std::string &textual)
```

Specifies an implicit value, which will be used if the option is given, but without an adjacent value. Using this implies that an explicit value is optional, but if given, must be strictly adjacent to the
option, i.e.: ‘-o value’ or ‘option=value’. Giving ‘-o’ or ‘option’ will cause the implicit value to be applied. Unlike the above overload, the type ‘T’ need not provide operator<< for ostream, but textual representation of default value must be provided by the user.

```cpp
typed_value *notifier (std::function<void>) const T&
    >/f Specifies a function to be called when the final value is determined.

typed_value *composing()
    Specifies that the value is composing. See the ‘is_composing’ method for explanation.

typed_value *multitoken()
    Specifies that the value can span multiple tokens.

typed_value *zero_tokens()
    Specifies that no tokens may be provided as the value of this option, which means that only presence of the option is significant. For such option to be useful, either the ‘validate’ function should be specialized, or the ‘implicit_value’ method should be also used. In most cases, you can use the ‘bool_switch’ function instead of using this method.

typed_value *required()
    Specifies that the value must occur.
```

```cpp
std::string name() const

bool is_composing() const

unsigned min_tokens() const

unsigned max_tokens() const

bool is_required() const

void xparse (hpx::util::any_nonser &value_store, const std::vector<std::basic_string<Char>> &new_tokens) const
    Creates an instance of the ‘validator’ class and calls its operator() to perform the actual conversion.

virtual bool apply_default (hpx::util::any_nonser &value_store) const
    If default value was specified via previous call to ‘default_value’, stores that value into ‘value_store’. Returns true if default value was stored.

void notify (const hpx::util::any_nonser &value_store) const
    If an address of variable to store value was specified when creating *this, stores the value there. Otherwise, does nothing.

const std::type_info &value_type() const
```

**Private Members**

```cpp
T *m_store_to
std::string m_value_name
hpx::util::any_nonser m_default_value
std::string m_default_value_as_text
hpx::util::any_nonser m_implicit_value
std::string m_implicit_value_as_text
bool m_composing
```
bool m_implicit
bool m_multitoken
bool m_zero_tokens
bool m_required

std::function<void (const T&) > m_notifier

class typed_value_base
#include <value_semantic.hpp> Base class for all option that have a fixed type, and are willing to announce this type to the outside world. Any ‘value_semantics’ for which you want to find out the type can be dynamic_cast-ed to typed_value_base. If conversion succeeds, the ‘type’ method can be called.

Subclassed by hpx::program_options::typed_value<T, Char>

Public Functions

virtual const std::type_info & value_type() const = 0
virtual ~typed_value_base()

class untyped_value : public hpx::program_options::value_semantic_codecvt_helper<char>
#include <value_semantic.hpp> Class which specifies a simple handling of a value: the value will have string type and only one token is allowed.

Public Functions

untyped_value(bool zero_tokens = false)

std::string name() const
Returns the name of the option. The name is only meaningful for automatic help message.

unsigned min_tokens() const
The minimum number of tokens for this option that should be present on the command line.

unsigned max_tokens() const
The maximum number of tokens for this option that should be present on the command line.

bool is_composing() const
Returns true if values from different sources should be composed. Otherwise, value from the first source is used and values from other sources are discarded.

bool is_required() const
Returns true if value must be given. Non-optional value

void xparse(hpx::util::any_nonser &value_store, const std::vector<std::string> &new_tokens) const
If ‘value_store’ is already initialized, or new_tokens has more than one elements, throws. Otherwise, assigns the first string from ‘new_tokens’ to ‘value_store’, without any modifications.

bool apply_default(hpx::util::any_nonser&) const
Does nothing.

void notify(const hpx::util::any_nonser&) const
Does nothing.
Private Members

bool m_zero_tokens

class value_semantic
#include <value_semantic.hpp> Class which specifies how the option’s value is to be parsed and converted into C++ types.

Subclassed by hpx::program_options::value_semantic_codecvt_helper< char >, hpx::program_options::value_semantic_codecvt_helper< wchar_t >

Public Functions

virtual std::string name() const = 0
Returns the name of the option. The name is only meaningful for automatic help message.

virtual unsigned min_tokens() const = 0
The minimum number of tokens for this option that should be present on the command line.

virtual unsigned max_tokens() const = 0
The maximum number of tokens for this option that should be present on the command line.

virtual bool is_composing() const = 0
Returns true if values from different sources should be composed. Otherwise, value from the first source is used and values from other sources are discarded.

virtual bool is_required() const = 0
Returns true if value must be given. Non-optional value

virtual void parse(hpx::util::any_nonser &value_store, const std::vector<std::string>& new_tokens, bool utf8) const = 0
 Parses a group of tokens that specify a value of option. Stores the result in ‘value_store’, using whatever representation is desired. May be be called several times if value of the same option is specified more than once.

virtual bool apply_default(hpx::util::any_nonser &value_store) const = 0
Called to assign default value to ‘value_store’. Returns true if default value is assigned, and false if no default value exists.

virtual void notify(const hpx::util::any_nonser &value_store) const = 0
Called when final value of an option is determined.

virtual ~value_semantic()

template <class Char>
class value_semantic_codecvt_helper
#include <value_semantic.hpp> Helper class which perform necessary character conversions in the ‘parse’ method and forwards the data further.

Subclassed by hpx::program_options::typed_value< T, Char >

template <>
template<>
class value_semantic_codecvt_helper< char >: public hpx::program_options::value_semantic
#include <value_semantic.hpp> Helper conversion class for values that accept ascii strings as input. Overrides the ‘parse’ method and defines new ‘xparse’ method taking std::string. Depending on whether input to parse is ascii or UTF8, will pass it to xparse unmodified, or with UTF8->ascii conversion.
Subclassed by `hpx::program_options::untyped_value`

### Protected Functions

```cpp
virtual void xparse(hpx::util::any_nonser &value_store, const std::vector<std::string> &new_tokens) const = 0
```

### Private Functions

```cpp
void parse(hpx::util::any_nonser &value_store, const std::vector<std::string> &new_tokens, bool utf8) const
```

Parses a group of tokens that specify a value of option. Stores the result in `value_store`, using whatever representation is desired. May be called several times if value of the same option is specified more than once.

```cpp
template <>
template<>

class value_semantic_codecvt_helper<wchar_t> : public hpx::program_options::value_semantic
#include <value_semantic.hpp>

Helper conversion class for values that accept ascii strings as input. Overrides the 'parse' method and defines new 'xparse' method taking std::wstring. Depending on whether input to parse is ascii or UTF8, will recode input to Unicode, or pass it unmodified.

### Protected Functions

```cpp
virtual void xparse(hpx::util::any_nonser &value_store, const std::vector<std::wstring> &new_tokens) const = 0
```

### Private Functions

```cpp
void parse(hpx::util::any_nonser &value_store, const std::vector<std::string> &new_tokens, bool utf8) const
```

Parses a group of tokens that specify a value of option. Stores the result in `value_store`, using whatever representation is desired. May be called several times if value of the same option is specified more than once.

```cpp
#include <hpx/program_options/option.hpp>
namespace hpx

namespace program_options

### Typedefs

```cpp
using hpx::program_options::option = typedef basic_option<char>
using hpx::program_options::woption = typedef basic_option<wchar_t>
```

```cpp
template <class Char>
class basic_option
#include <option.hpp>

Option found in input source. Contains a key and a value. The key, in turn, can be a string (name of an option), or an integer (position in input source) – in case no name is specified. The latter is only possible for command line. The template parameter specifies the type of char used for storing the option’s value.
Public Functions

basic_option()

basic_option(const std::string &xstring_key, const std::vector<std::string> &xvalue)

Public Members

std::string string_key
String key of this option. Intentionally independent of the template parameter.

int position_key
Position key of this option. All options without an explicit name are sequentially numbered starting from 0. If an option has explicit name, ‘position_key’ is equal to -1. It is possible that both position_key and string_key is specified, in case name is implicitly added.

std::vector<std::basic_string<Char>> value
Option’s value

std::vector<std::basic_string<Char>> original_tokens
The original unchanged tokens this option was created from.

bool unregistered
True if option was not recognized. In that case, ‘string_key’ and ‘value’ are results of purely syntactic parsing of source. The original tokens can be recovered from the “original_tokens” member.

bool case_insensitive
True if string_key has to be handled case insensitive.

#include <hpx/program_options/cmdline.hpp>

namespace hpx

namespace program_options

namespace command_line_style

Enums

enum style_t
Various possible styles of options.

There are “long” options, which start with “–” and “short”, which start with either “-” or “/”. Both kinds can be allowed or disallowed, see allow_long and allow_short. The allowed character for short options is also configurable.

Option’s value can be specified in the same token as name (“-foo=bar”), or in the next token.

It’s possible to introduce long options by the same character as short options, see allow_long_disguise.

Finally, guessing (specifying only prefix of option) and case insensitive processing are supported.

Values:

allow_long = 1
Allow “–long_name” style.
allow_short = allow_long << 1
Allow "-<single character" style.

allow_dash_for_short = allow_short << 1
Allow "-" in short options.

allow_slash_for_short = allow_dash_for_short << 1
Allow "/" in short options.

long_allow_adjacent = allow_slash_for_short << 1
Allow option parameter in the same token for long option, like in

```text
--foo=10
```

long_allow_next = long_allow_adjacent << 1
Allow option parameter in the next token for long options.

short_allow_adjacent = long_allow_next << 1
Allow option parameter in the same token for short options.

short_allow_next = short_allow_adjacent << 1
Allow option parameter in the next token for short options.

allow_sticky = short_allow_next << 1
Allow to merge several short options together, so that "-s -k" become "-sk". All of the options but last should accept no parameter. For example, if "-s" accept a parameter, then "k" will be taken as parameter, not another short option. Dos-style short options cannot be sticky.

allow_guessing = allow_sticky << 1
Allow abbreviated spellings for long options, if they unambiguously identify long option. No long option name should be prefix of other long option name if guessing is in effect.

long_case_insensitive = allow_guessing << 1
Ignore the difference in case for long options.

short_case_insensitive = long_case_insensitive << 1
Ignore the difference in case for short options.

case_insensitive = (long_case_insensitive | short_case_insensitive)
Ignore the difference in case for all options.

allow_long_disguise = short_case_insensitive << 1
Allow long options with single option starting character, e.g -foo=10

allow_long | long_allow_adjacent | long_allow_next | allow_sticky | allow_guessing | allow_dash_for_short) ]The more-or-less traditional unix style.

default_style = unix_style
The default style.

hashing

#include <compatibility/hpx/util/fibhash.hpp>

#include <compatibility/hpx/util/jenkins_hash.hpp>

#include <hpx/hashing/fibhash.hpp>

namespace hpx

    namespace util
Functions

```cpp
#include <hpx/hashing/jenkins_hash.hpp>
namespace hpx
namespace util

class jenkins_hash
#include <jenkins_hash.hpp> The jenkins_hash class encapsulates a hash calculation function published by Bob Jenkins here: http://burtleburtle.net/bob/hash

Public Types

enum seedenum
The seedenum is used as a dummy parameter to distinguish the different constructors

Values:
seed = 1

typedef std::uint32_t size_type
this is the type representing the result of this hash

Public Functions

jenkins_hash()
constructors and destructor

jenkins_hash(size_type size)

jenkins_hash(size_type seedval, seedenum)

~jenkins_hash()

size_type operator()(std::string const &key) const
calculate the hash value for the given key

size_type operator()(char const *key) const

bool reset(size_type size)
re-seed the hash generator

void set_seed(size_type seedval)
initialize the hash generator to a specific seed

void swap(jenkins_hash &rhs)
support for std::swap

Protected Functions

size_type hash(const char *k, std::size_t length) const
```
Private Members

\texttt{size\_type seed\_}

\texttt{config}

\texttt{#include <hpx/config.hpp>}

\texttt{Defines}

\texttt{HPX\_INITIAL\_IP\_PORT}
This is the default ip/port number used by the parcel subsystem.

\texttt{HPX\_CONNECTING\_IP\_PORT}

\texttt{HPX\_INITIAL\_IP\_ADDRESS}

\texttt{HPX\_RUNTIME\_INSTANCE\_LIMIT}
This defines the maximum number of possible runtime instances in one executable

\texttt{HPX\_PARCEL\_BOOTSTRAP}
This defines the type of the parcelport to be used during application bootstrap. This value can be changed at runtime by the configuration parameter:

\begin{verbatim}
    hpx.parcel.bootstrap = ...
\end{verbatim}

(or by setting the corresponding environment variable \texttt{HPX\_PARCEL\_BOOTSTRAP}).

\texttt{HPX\_PARCEL\_MAX\_CONNECTIONS}
This defines the number of outgoing (parcel-) connections kept alive (to all other localities). This value can be changed at runtime by setting the configuration parameter:

\begin{verbatim}
    hpx.parcel.max_connections = ...
\end{verbatim}

(or by setting the corresponding environment variable \texttt{HPX\_PARCEL\_MAX\_CONNECTIONS}).

\texttt{HPX\_PARCEL\_IPC\_DATA\_BUFFER\_CACHE\_SIZE}
This defines the number of outgoing ipc (parcel-) connections kept alive (to each of the other localities on the same node). This value can be changed at runtime by setting the configuration parameter:

\begin{verbatim}
    hpx.parcel.ipc.data_buffer_cache_size = ...
\end{verbatim}

(or by setting the corresponding environment variable \texttt{HPX\_PARCEL\_IPC\_DATA\_BUFFER\_CACHE\_SIZE}).

\texttt{HPX\_PARCEL\_MPI\_MAX\_REQUESTS}
This defines the number of MPI requests in flight This value can be changed at runtime by setting the configuration parameter:

\begin{verbatim}
    hpx.parcel.mpi.max_requests = ...
\end{verbatim}

(or by setting the corresponding environment variable \texttt{HPX\_PARCEL\_MPI\_MAX\_REQUESTS}).

\texttt{HPX\_PARCEL\_MAX\_CONNECTIONS\_PER\_LOCALITY}
This defines the number of outgoing (parcel-) connections kept alive (to each of the other localities). This value can be changed at runtime by setting the configuration parameter:

\begin{verbatim}
    hpx.parcel.max_connections_per_locality = ...
\end{verbatim}

(or by setting the corresponding environment variable \texttt{HPX\_PARCEL\_MAX\_CONNECTIONS\_PER\_LOCALITY}).

\texttt{HPX\_PARCEL\_MAX\_MESSAGE\_SIZE}
This defines the maximally allowed message size for messages transferred between localities. This value can be changed at runtime by setting the configuration parameter:

\begin{verbatim}
    hpx.parcel.max_message_size = ...
\end{verbatim}
(or by setting the corresponding environment variable HPX_PARCEL_MAX_MESSAGE_SIZE).

**HPX_PARCEL_MAX_OUTBOUND_MESSAGE_SIZE**
This defines the maximally allowed outbound message size for coalescing messages transferred between locali-
ties. This value can be changed at runtime by setting the configuration parameter:

```cpp
hpx.parcel.max_outbound_message_size = ...
```

(or by setting the corresponding environment variable HPX_PARCEL_MAX_OUTBOUND_MESSAGE_SIZE).

**HPX_PARCEL_SERIALIZATION_OVERHEAD**

**HPX_AGAS_LOCAL_CACHE_SIZE**
This defines the number of AGAS address translations kept in the local cache. This is just the initial size which
may be adjusted depending on the load of the system (not implemented yet), etc. It must be a minimum of 3 for
AGAS v3 bootstrapping.

This value can be changes at runtime by setting the configuration parameter:

```cpp
hpx.agas.local_cache_size = ...
```

(or by setting the corresponding environment variable HPX_AGAS_LOCAL_CACHE_SIZE)

**HPX_INITIAL_AGAS_MAX_PENDING_REFCNT_REQUESTS**

**HPX_GLOBALCREDIT_INITIAL**
This defines the initial global reference count associated with any created object.

**HPX_NUM_IO_POOL_SIZE**
This defines the default number of OS-threads created for the different internal thread pools

**HPX_NUM_PARCEL_POOL_SIZE**

**HPX_NUM_TIMER_POOL_SIZE**

**HPX_SPINLOCK_DEADLOCK_DETECTION_LIMIT**
By default, enable minimal thread deadlock detection in debug builds only.

**HPX_COROUTINE_NUM.HEAPS**
This defines the default number of coroutine heaps.

**HPX_HAVE_THREAD_BACKTRACE_DEPTH**
By default, enable storing the thread phase in debug builds only.

By default, enable storing the parent thread information in debug builds only. By default, enable storing the
thread description in debug builds only. By default, enable storing the target address of the data the thread is
accessing in debug builds only. By default we do not maintain stack back-traces on suspension. This is a pure
debugging aid to be able to see in the debugger where a suspended thread got stuck. By default we capture only
5 levels of stack back trace on suspension

**HPX_MAX_NETWORK_RETRIES**

**HPX_NETWORK_RETRIES_SLEEP**

**HPX_INI_PATH_DELIMITER**

**HPX_PATH_DELIMITERS**

**HPX_SHARED_LIB_EXTENSION**

**HPX_EXECUTABLE_EXTENSION**

**HPX_MAKE_DLL_STRING**

**HPX_MANGLE_NAME**

**HPX_MANGLE_STRING**
```cpp
#include <hpx/config/lambda_capture.hpp>

Defines

HPX_CAPTURE_FORWARD(var)
    Evaluates to \texttt{var = \texttt{std::forward<decltype(var)>(var)}} if the compiler supports C++14 Lambdas.
    Defaults to \texttt{var}.
```
`HPX_CAPTURE_MOVE(var)`
Evaluates to `var = std::move(var)` if the compiler supports C++14 Lambdas. Defaults to `var`.

```cpp
#include <hpx/config/attributes.hpp>
```
Defines

`HPX_NOINLINE`
Function attribute to tell compiler not to inline the function.

`HPX_NORETURN`
Function attribute to tell compiler that the function does not return.

`HPX_DEPRECATED(x)`
Marks an entity as deprecated. The argument `x` specifies a custom message that is included in the compiler warning. For more details see `<>`.

`HPX_FALLTHROUGH`
Indicates that the fall through from the previous case label is intentional and should not be diagnosed by a compiler that warns on fallthrough. For more details see `<>`.

```cpp
#include <hpx/config/forceinline.hpp>
```
Defines

`HPX_FORCEINLINE`
Marks a function to be forced inline.

```cpp
#include <hpx/config/warnings_prefix.hpp>
#include <hpx/config/warnings_suffix.hpp>
#include <hpx/config/constexpr.hpp>
```
Defines

`HPX_CONSTEXPR`
This macro evaluates to `constexpr` if the compiler supports it.

`HPX_CONSTEXPR_OR_CONST`
This macro evaluates to `constexpr` if the compiler supports it, `const` otherwise.

`HPX_CXX14_CONSTEXPR`
This macro evaluates to `constexpr` if the compiler supports C++14 constexpr.

`HPX_STATIC_CONSTEXPR`
This macro evaluates to `static :c:macro:HPX_CONSTEXPR_OR_CONST`.

```cpp
#include <hpx/config/debug.hpp>
```
Defines

`HPX_DEBUG`
Defined if HPX is compiled in debug mode.

`HPX_BUILD_TYPE`
Evaluates to `debug` if compiled in debug mode, `release` otherwise.
#include <hpx/config/manual_profiling.hpp>

Defines

HPX_SUPER_PURE
HPX_PURE
HPX_HOT
HPX_COLD

#include <hpx/config/emulate_deleted.hpp>

Defines

HPX_NON_COPYABLE(cls)
   Marks a class as non-copyable and non-moveable.

#include <hpx/config/force_linking.hpp>

namespace hpx

    namespace config

        Functions

        force_linking_helper &force_linking()
        struct force_linking_helper

            Public Members

            const char *const hpx_version
            const char *const boost_version

#include <hpx/config/weak_symbol.hpp>

Defines

HPX_WEAK_SYMBOL

#include <hpx/config/branch_hints.hpp>

Defines

HPX_LIKELY(expr)
   Hint at the compiler that expr is likely to be true.

HPX_UNLIKELY(expr)
   Hint at the compiler that expr is likely to be false.
#include <hpx/config/compiler_fence.hpp>

Defines

HPX_COMPILER_FENCE
Generates assembly that serves as a fence to the compiler CPU to disable optimization. Usually implemented in the form of a memory barrier.

HPX_SMT_PAUSE
Generates assembly that executes a “pause” instruction. Useful in spinning loops.

#include <hpx/config/compiler_native_tls.hpp>

Defines

HPX_NATIVE_TLS
This macro is replaced with the compiler specific keyword attribute to mark a variable as thread local. For more details see `<__`

#include <hpx/config/threads_stack.hpp>

Defines

HPX_THREADS_STACK_OVERHEAD
HPX_SMALL_STACK_SIZE
HPX_MEDIUM_STACK_SIZE
HPX_LARGE_STACK_SIZE
HPX_HUGE_STACK_SIZE

#include <hpx/config/export_definitions.hpp>

Defines

HPX_EXPORT
Marks a class or function to be exported from HPX or imported if it is consumed.

#include <hpx/config/compiler_specific.hpp>

Defines

HPX_GCC_VERSION
Returns the GCC version HPX is compiled with. Only set if compiled with GCC.

HPX_CLANG_VERSION
Returns the Clang version HPX is compiled with. Only set if compiled with Clang.

HPX_INTEL_VERSION
Returns the Intel Compiler version HPX is compiled with. Only set if compiled with the Intel Compiler.

HPX_MSVC
This macro is set if the compilation is with MSVC.

HPX_MINGW
This macro is set if the compilation is with Mingw.
HPX WINDOWS
This macro is set if the compilation is for Windows.

HPX_NATIVE_MIC
This macro is set if the compilation is for Intel Knights Landing.

#include <hpx/config/autolink.hpp>
#include <hpx/config/asio.hpp>

iterator_support
#include <compatibility/hpx/util/transform_iterator.hpp>
#include <compatibility/hpx/util/range.hpp>
#include <compatibility/hpx/util/iterator_facade.hpp>
#include <compatibility/hpx/util/iterator_adaptor.hpp>
#include <compatibility/hpx/util/iterator_range.hpp>
#include <compatibility/hpx/traits/is_range.hpp>
#include <compatibility/hpx/traits/is_iterator.hpp>
#include <hpx/iterator_support/transform_iterator.hpp>

namespace hpx

namespace util

Functions

template <typename Transformer, typename Iterator>
transform_iterator<Iterator, Transformer> make_transform_iterator (Iterator const &it,
  Transformer const &f)

template <typename Transformer, typename Iterator>
transform_iterator<Iterator, Transformer> make_transform_iterator (Iterator const &it)

template <typename Iterator, typename Transformer, typename Reference, typename Value, typename Category, typename Difference>

Public Functions

transform_iterator ()
transform_iterator (Iterator const &it)
transform_iterator (Iterator const &it, Transformer const &f)

template <typename OtherIterator, typename OtherTransformer, typename OtherReference, typename OtherValue, typename OtherCategory, typename OtherDifference>
transform_iterator\( (\texttt{transform\_iterator}\langle \text{OtherIterator}, \text{OtherTransformer}, \text{OtherReference}, \text{OtherValue}, \text{OtherCategory}, \text{OtherDifference}\rangle \text{ const} \& t, \text{typename std::enable\_if<}\langle \text{std::is\_convertible<}\langle \text{OtherIterator}, \text{Iterator}\rangle::\text{value} \&\& \text{std::is\_convertible<}\langle \text{OtherTransformer}, \text{Transformer}\rangle::\text{value} \&\& \text{std::is\_convertible<}\langle \text{OtherCategory}, \text{Category}\rangle::\text{value} \&\& \text{std::is\_convertible<}\langle \text{OtherDifference}, \text{Difference}\rangle::\text{value} \rangle::\text{type} * = \text{nullptr} \rangle \text{&& std::is\_convertible<}\langle \text{OtherIterator}, \text{Iterator}\rangle::\text{value} \&\& \text{std::is\_convertible<}\langle \text{OtherTransformer}, \text{Transformer}\rangle::\text{value} \&\& \text{std::is\_convertible<}\langle \text{OtherCategory}, \text{Category}\rangle::\text{value} \&\& \text{std::is\_convertible<}\langle \text{OtherDifference}, \text{Difference}\rangle::\text{value} \rangle::\text{type} * = \text{nullptr} \rangle
\)

Transformer \text{const} \& \text{transformer} () \text{ const}

Private Types

typedef detail::transform\_iterator\_base\langle \text{Iterator}, \text{Transformer}, \text{Reference}, \text{Value}, \text{Category}, \text{Difference}\rangle::\text{type base}

Private Functions

\text{base\_type}::\text{reference} \text{dereference} () \text{ const}

Private Members

Transformer \text{transformer}_

Friends

friend \text{hpx::util::hpx::util::iterator\_core\_access}

#include <hpx/iterator\_support/range.hpp>

namespace \text{hpx}

namespace \text{util}

namespace \text{range\_adl}

Functions

\text{template <typename C, typename \text{typename \text{Iterator} = typename detail::iterator<}\langle C\rangle::\text{type}}}\\ \text{HPX\_HOST\_DEVICE \text{Iterator hpx::util::range\_adl::begin}(C \& c)}

\text{template <typename C, typename \text{typename \text{Iterator} = typename detail::iterator<}\langle C \text{ const}\rangle::\text{type}}}\\ \text{HPX\_HOST\_DEVICE \text{Iterator hpx::util::range\_adl::begin}(C \text{ const} \& c)}

\text{template <typename C, typename \text{typename \text{Sentinel} = typename detail::sentinel<}\langle C\rangle::\text{type}}}\\ \text{HPX\_HOST\_DEVICE \text{Sentinel hpx::util::range\_adl::end}(C \& c)}

\text{template <typename C, typename \text{typename \text{Sentinel} = typename detail::sentinel<}\langle C \text{ const}\rangle::\text{type}}}\\ \text{HPX\_HOST\_DEVICE \text{Sentinel hpx::util::range\_adl::end}(C \text{ const} \& c)}

\text{template <typename C, typename \text{typename \text{typename} \text{Sentinel} = \text{typename}}}\\ \text{HPX\_HOST\_DEVICE std::size\_t hpx::util::range\_adl::size(C \text{ const} \& c)}

\text{template <typename C, typename \text{typename \text{typename} \text{Sentinel} = \text{typename}}}\\ \text{HPX\_HOST\_DEVICE bool hpx::util::range\_adl::empty(C \text{ const} \& c)}

2.9. API reference 595
#include <hpx/iterator_support/iterator_facade.hpp>

## Defines

HPX_UTIL_ITERATOR_FACADE_INTEROP_HEAD (prefix, op, result_type)

namespace hpx

namespace util

### Functions

```cpp
template <typename Derived, typename T, typename Category, typename Reference, typename Distance, typename Pointer>
HPX_HOST_DEVICE util::detail::postfix_increment_result<Derived, typename Derived::value_type, typename Derived::reference>::type hpx::util::operator++(iterator_facade < Derived, T, Category, Reference, Distance, Pointer > & i, int)
```

```cpp
hpx::util::HPX_UTIL_ITERATOR_FACADE_INTEROP_HEAD(inline, bool)
```

```cpp
hpx::util::HPX_UTIL_ITERATOR_FACADE_INTEROP_HEAD(inline, !, bool)
```

```cpp
hpx::util::HPX_UTIL_ITERATOR_FACADE_INTEROP_HEAD(inline)
```

```cpp
hpx::util::HPX_UTIL_ITERATOR_FACADE_INTEROP_HEAD(inline, <=, bool)
```

```cpp
hpx::util::HPX_UTIL_ITERATOR_FACADE_INTEROP_HEAD(inline, >=, bool)
```

```cpp
hpx::util::HPX_UTIL_ITERATOR_FACADE_INTEROP_HEAD(inline, -, typename std::iterator_traits< Derived >::difference_type)
```

```cpp
template <typename Derived, typename T, typename Category, typename Reference, typename Distance, typename Pointer>
HPX_HOST_DEVICE Derived hpx::util::operator+(iterator_facade < Derived, T, Category, Reference, Distance, Pointer > const & it, typename Derived::difference_type n)
```

```cpp
template <typename Derived, typename T, typename Category, typename Reference, typename Distance, typename Pointer>
HPX_HOST_DEVICE Derived hpx::util::operator+(typename Derived::difference_type n, iterator_facade < Derived, T, Category, Reference, Distance, Pointer > const & it)
```

### Public Static Functions

```cpp
template <typename Iterator1, typename Iterator2>
static HPX_HOST_DEVICE bool hpx::util::iterator_core_access::equal(Iterator1 const & lhs, Iterator2 const & rhs)
```

```cpp
template <typename Iterator>
static HPX_HOST_DEVICE void hpx::util::iterator_core_access::increment(Iterator & it)
```

```cpp
template <typename Iterator>
static HPX_HOST_DEVICE void hpx::util::iterator_core_access::decrement(Iterator & it)
```

```cpp
template <typename Reference, typename Iterator>
static HPX_HOST_DEVICE Reference hpx::util::iterator_core_access::dereference(Iterator const & it)
```

```cpp
template <typename Iterator, typename Distance>
static HPX_HOST_DEVICE void hpx::util::iterator_core_access::advance(Iterator & it, Distance n)
```

```cpp
template <typename Iterator1, typename Iterator2>
static HPX_HOST_DEVICE std::iterator_traits<Iterator1>::difference_type hpx::util::iterator_core_access::distance_to(Iterator1 const & lhs, Iterator2 const & rhs)
```

### Subclassed by

hpx::util::iterator_adaptor< Derived, Base, Value, Category, Reference, Difference, Pointer >
Public Functions

HPX_HOST_DEVICE iterator_facade()

Protected Types

typedef iterator_facade<Derived, T, Category, Reference, Distance, Pointer> iterator_adaptor_

Private Types

typedef detail::iterator_facade_base<Derived, T, Category, Reference, Distance, Pointer> base_type

#include <hpx/iterator_support/iterator_adaptor.hpp>

namespace hpx

namespace util

template <typename Derived, typename Base, typename Value = void, typename Category = void, typename Reference = void, typename Difference = void, typename Pointer = void>
class iterator_adaptor: public hpx::util::iterator_facade<Derived, value_type, iterator_category, reference_type, difference_type, Pointer>

Subclassed by hpx::util::transform_iterator< Iterator, Transformer, Reference, Value, Category, Difference >

Public Types

typedef Base base_type

Public Functions

HPX_HOST_DEVICE iterator_adaptor()

HPX_HOST_DEVICE iterator_adaptor (Base const &iter)

HPX_HOST_DEVICE Base const& hpx::util::iterator_adaptor::base() const

Protected Types

typedef hpx::util::detail::iterator_adaptor_base<Derived, Base, Value, Category, Reference, Difference, Pointer>::type iterator_adaptor_

Protected Functions

HPX_HOST_DEVICE Base const& hpx::util::iterator_adaptor::base_reference() const

HPX_HOST_DEVICE Base& hpx::util::iterator_adaptor::base_reference()

Private Functions

HPX_HOST_DEVICE base_adaptor_type::reference hpx::util::iterator_adaptor::dereference() const

template <typename OtherDerived, typename OtherIterator, typename V, typename C, typename R, typename D, typename P>
HPX_HOST_DEVICE bool hpx::util::iterator_adaptor::equal(iterator_adaptor < OtherDerived, OtherIterator, V, C, R, D, P > const & x) const

template <typename DifferenceType>
Private Members

Base iterator_

Friends

friend hpx::util::hpx::util::iterator_core_access

#include <hpx/iterator_support/iterator_range.hpp>

namespace hpx

namespace util

Functions

template <typename Range, typename Iterator = typename traits::range_iterator<Range>::type, typename Sentinel = typename traits::range_iterator<Range>::type>
std::enable_if<traits::is_range<Range>::value, iterator_range<Iterator, Sentinel>>::type make_iterator_range(Range & r)

template <typename Range, typename Iterator = typename traits::range_iterator<Range const>::type, typename Sentinel = typename traits::range_iterator<Range const>::type>
std::enable_if<traits::is_range<Range>::value, iterator_range<Iterator, Sentinel>>::type make_iterator_range(Range const & r)

template <typename Iterator, typename Sentinel = Iterator>
std::enable_if<traits::is_iterator<Iterator>::value, iterator_range<Iterator, Sentinel>>::type make_iterator_range(Iterator iterator, Sentinel sentinel)

template <typename Iterator, typename Sentinel = Iterator>
class iterator_range

Public Functions

iterator_range ()

iterator_range (Iterator iterator, Sentinel sentinel)

Iterator begin () const
Iterator `end() const`  
`std::ptrdiff_t size() const`  
bool `empty() const`  

**Private Members**  
Iterator `_iterator`  
Sentinel `_sentinel`  

```cpp
#include <hpx/iterator_support/zip_iterator.hpp>

template <typename F, typename... Ts>
struct lift_zipped_iterators<F, util::zip_iterator<Ts...>>

Public Types

typedef util::zip_iterator<Ts...>::iterator_tuple_type tuple_type

typedef util::tuple<
    typename element_result_of<
        typename F::template apply<Ts>, Ts>::type...
    > result_type

Public Static Functions

template <std::size_t..., Is, typename... Ts>
static result_type call (util::detail::pack_c<
    std::size_t, Is...>, util::tuple<Ts...> const &t)

template <typename... Ts>
static result_type call (util::zip_iterator<Ts...> const &iter)

namespace hpx

namespace traits

namespace functional

```
```
namespace util

Functions

```cpp
template <typename... Ts>
HPX_HOST_DEVICE zip_iterator<typename decay<Ts>::type...> hpx::util::make_zip_iterator(Ts &&... vs)

template <typename... Ts>
class zip_iterator : public hpx::util::detail::zip_iterator_base<tuple<Ts...>, zip_iterator<Ts...>>
```

Public Functions

```cpp
HPX_HOST_DEVICE zip_iterator ()
HPX_HOST_DEVICE zip_iterator (Ts const&... vs)
HPX_HOST_DEVICE zip_iterator (tuple<Ts...> &&t)
HPX_HOST_DEVICE zip_iterator (zip_iterator const & other)
HPX_HOST_DEVICE zip_iterator (zip_iterator && other)
```

```cpp
template <typename... Ts_>
HPX_HOST_DEVICE std::enable_if< detail::are_tuples_compatible_not_same< typename zip_iterator::iterator_tuple_type, ... zip_iterator&>::type hpx::util::zip_iterator::operator=(zip_iterator < Ts_... > const & other)
template <typename... Ts_>
HPX_HOST_DEVICE std::enable_if< detail::are_tuples_compatible_not_same< typename zip_iterator::iterator_tuple_type, ... zip_iterator&>::type hpx::util::zip_iterator::operator=(zip_iterator < Ts_... > && other)
```

Private Types

```cpp
typedef detail::zip_iterator_base<tuple<Ts...>, zip_iterator<Ts...>> base_type
```

```cpp
template <typename... Ts>
template<>
class zip_iterator<tuple<Ts...>> : public hpx::util::detail::zip_iterator_base<tuple<Ts...>, zip_iterator<Ts...>>
```

Public Functions

```cpp
HPX_HOST_DEVICE zip_iterator ()
HPX_HOST_DEVICE zip_iterator (Ts const&... vs)
HPX_HOST_DEVICE zip_iterator (tuple<Ts...> &&t)
HPX_HOST_DEVICE zip_iterator (zip_iterator const & other)
HPX_HOST_DEVICE zip_iterator (zip_iterator && other)
```

```cpp
template <typename... Ts_>
HPX_HOST_DEVICE std::enable_if< detail::are_tuples_compatible_not_same< typename zip_iterator::iterator_tuple_type, ... zip_iterator&>::type hpx::util::zip_iterator::operator=(zip_iterator < Ts_... > const & other)
template <typename... Ts_>
HPX_HOST_DEVICE std::enable_if< detail::are_tuples_compatible_not_same< typename zip_iterator::iterator_tuple_type, ... zip_iterator&>::type hpx::util::zip_iterator::operator=(zip_iterator < Ts_... > && other)
```

600 Chapter 2. What’s so special about HPX?
Private Types

typedef detail::zip_iterator_base<
tuple<Ts...>, zip_iterator<
tuple<Ts...>>> base_type

#include <hpx/iterator_support/traits/is_range.hpp>

namespace hpx

namespace traits

    template <typename R>
    template<>
    struct range_traits<R, true> : public std::iterator_traits<
    util::detail::iterator<R>::type>

Public Types

typedef util::detail::iterator<R>::type iterator_type

typedef util::detail::sentinel<R>::type sentinel_type

#include <hpx/iterator_support/traits/is_iterator.hpp>

format

#include <compatibility/hpx/util/format.hpp>

#include <hpx/format.hpp>

Defines

DECL_TYPE_SPECIFIER (Type, Spec)

HPX_FORMAT_EXPORT

namespace hpx

namespace util

Functions

    template <typename... Args>
    std::string format (boost::string_ref format_str, Args const&... args)

    template <typename... Args>
    std::ostream &format_to (std::ostream &os, boost::string_ref format_str, Args const&... args)

concepts

#include <compatibility/hpx/traits/has_XXX.hpp>

#include <compatibility/hpx/traits/concepts.hpp>

#include <compatibility/hpx/traits/has_member_XXX.hpp>

#include <hpx/concepts/has_XXX.hpp>
Defines

HPX_HAS_XXX_TRAIT_DEF (Name)

This macro creates a boolean unary metafunction such that for any type X, has_name<X>::value == true if and only if X is a class type and has a nested type member x::name. The generated trait ends up in a namespace where the macro itself has been placed.

#include <hpx/concepts/concepts.hpp>

Defines

HPX_CONCEPT_REQUIRES(_,...)
HPX_CONCEPT_REQUIRES(...)
HPX_CONCEPT_ASSERT(...) 

#include <hpx/concepts/has_member_XXX.hpp>

Defines

HPX_HAS_MEMBER_XXX_TRAIT_DEF (MEMBER)

This macro creates a boolean unary metafunction which result is true if and only if its parameter type has member function with MEMBER name (no matter static it is or not). The generated trait ends up in a namespace where the macro itself has been placed.

allocator_support

#include <compatibility/hpx/util/allocator_deleter.hpp>
#include <compatibility/hpx/util/internal_allocator.hpp>
#include <hpx/allocator_support/allocator_deleter.hpp>

namespace hpx

namespace util

    template <typename Allocator>
    struct allocator_deleter

    Public Functions

    template <typename SharedState>
    void operator() (SharedState *state)

    Public Members

    Allocator alloc_

#include <hpx/allocator_support/internal_allocator.hpp>

namespace hpx

namespace util
Typedefs

```cpp
using hpx::util::internal_allocator = typedef std::allocator<T>
```

segmented_algorithms

```cpp
#include <hpx/parallel/segmented_algorithm.hpp>
#include <hpx/parallel/segmented_algorithms/reduce.hpp>
#include <hpx/parallel/segmented_algorithms/all_any_none.hpp>
#include <hpx/parallel/segmented_algorithms/find.hpp>
#include <hpx/parallel/segmented_algorithms/for_each.hpp>
#include <hpx/parallel/segmented_algorithms/fill.hpp>
#include <hpx/parallel/segmented_algorithms/transform_inclusive_scan.hpp>
#include <hpx/parallel/segmented_algorithms/adjacent_find.hpp>
#include <hpx/parallel/segmented_algorithms/minmax.hpp>
#include <hpx/parallel/segmented_algorithms/transform.hpp>
#include <hpx/parallel/segmented_algorithms/exclusive_scan.hpp>
#include <hpx/parallel/segmented_algorithms/generate.hpp>
#include <hpx/parallel/segmented_algorithms/inclusive_scan.hpp>
#include <hpx/parallel/segmented_algorithms/adjacent_difference.hpp>
#include <hpx/parallel/segmented_algorithms/transform_exclusive_scan.hpp>
#include <hpx/parallel/segmented_algorithms/transform_reduce.hpp>
#include <hpx/parallel/segmented_algorithms/count.hpp>
```

util

```cpp
#include <hpx/util/insert_checked.hpp>
```

namespace hpx

```cpp
namespace util

Functions

```cpp
template <typename Iterator>
bool insert_checked (std::pair<Iterator, bool> const &r)
```

Helper function for writing predicates that test whether an std::map insertion succeeded. This inline template function negates the need to explicitly write the sometimes lengthy std::pair<Iterator, bool> type.

```cpp
Return  This function returns r.second.
Parameters
```

2.9. API reference 603
r: [in] The return value of a std::map insert operation.

```cpp
template <typename Iterator>
bool insert_checked (std::pair<Iterator, bool> const & r, Iterator & it)
```

Helper function for writing predicates that test whether an std::map insertion succeeded. This inline template function negates the need to explicitly write the sometimes lengthy std::pair<Iterator, bool> type.

**Return** This function returns `r.second`.

**Parameters**
- `r`: [in] The return value of a std::map insert operation.
- `r`: [out] A reference to an Iterator, which is set to `r.first`.

```cpp
#include <hpx/util/calculate_fanout.hpp>

namespace hpx

namespace util

Functions

```cpp
std::size_t calculate_fanout (std::size_t size, std::size_t local_fanout)
```

```cpp
#include <hpx/util/await_traits.hpp>
#include <hpx/util/get_and_reset_value.hpp>

namespace hpx

namespace util

Functions

```cpp
std::uint64_t get_and_reset_value (std::uint64_t & value, bool reset)
std::int64_t get_and_reset_value (std::int64_t & value, bool reset)
```

```cpp
template <typename T>
T get_and_reset_value (std::atomic<T> & value, bool reset)
```

```cpp
std::vector<std::int64_t> get_and_reset_value (std::vector<std::int64_t> & value, bool reset)
```

```cpp
#include <hpx/util/manage_config.hpp>

namespace hpx

namespace util

struct manage_config

Public Types

```cpp
typedef std::map<std::string, std::string> map_type
```
Public Functions

```cpp
manage_config (std::vector<std::string> const &cfg)
void add (std::vector<std::string> const &cfg)
``` 

```cpp
template <typename T>
T get_value (std::string const &key, T dflt = T()) const
``` 

Public Members

```cpp
map_type config_
``` 

```cpp
#include <hpx/util/safe_lexical_cast.hpp>
namespace hpx
``` 

```cpp
namespace util
``` 

Functions

```cpp
template <typename DestType, typename SrcType>
DestType safe_lexical_cast (SrcType const &value, DestType const &dflt = DestType())
``` 

```cpp
template <typename DestType, typename Config>
std::enable_if<std::is_integral<DestType>::value, DestType>::type
get_entry_as (Config const &config, std::string const &key, DestType const &dflt)
``` 

```cpp
template <typename DestType, typename Config>
DestType get_entry_as (Config const &config, std::string const &key, std::string const &dflt)
``` 

synchronization

```cpp
#include <compatibility/hpx/lcos/local/event.hpp>
#include <compatibility/hpx/lcos/local/mutex.hpp>
#include <compatibility/hpx/lcos/local/spinlock_no_backoff.hpp>
#include <compatibility/hpx/lcos/local/counting_semaphore.hpp>
#include <compatibility/hpx/lcos/local/recursive_mutex.hpp>
#include <compatibility/hpx/lcos/local/no_mutex.hpp>
#include <compatibility/hpx/lcos/local/shared_mutex.hpp>
#include <compatibility/hpx/lcos/local/once.hpp>
```
Event semaphores can be used for synchronizing multiple threads that need to wait for an event to occur. When the event occurs, all threads waiting for the event are woken up.

Public Functions

**event()**
Construct a new event semaphore.

**bool occurred()**
Check if the event has occurred.

**void wait()**
Wait for the event to occur.

**void set()**
Release all threads waiting on this semaphore.

**void reset()**
Reset the event.

Private Types

**typedef lcos::local::spinlock mutex_type**

Private Functions

**void wait_locked(std::unique_lock<mutex_type> &l)**

**void set_locked(std::unique_lock<mutex_type> l)**
Private Members

mutex_type mtx_
  This mutex protects the queue.
local::detail::condition_variable cond_
std::atomic<bool> event_

#include <hpx/synchronization/mutex.hpp>

namespace hpx

namespace lcos

namespace local

class mutex
  Subclassed by hpx::lcos::local::timed_mutex

Public Functions

HPX_NON_COPYABLE (mutex)
mutex (char const *const description = "")
mutex ()
void lock (char const *const description, error_code &ec = throws)
void lock (error_code &ec = throws)
bool try_lock (char const *description, error_code &ec = throws)
bool try_lock (error_code &ec = throws)
void unlock (error_code &ec = throws)

Protected Types

typedef lcos::local::spinlock mutex_type

Protected Attributes

mutex_type mtx_
  threads::thread_id_type owner_id_
  detail::condition_variable cond_

class timed_mutex : hpx::lcos::local::mutex
Public Functions

HPX\_NON\_COPYABLE (timed\_mutex)

timed\_mutex (char const *const description = "")

~timed\_mutex ()

bool try\_lock\_until (util::steady\_time\_point const &abs\_time, char const *description, error\_code &ec = throws)

bool try\_lock\_until (util::steady\_time\_point const &abs\_time, error\_code &ec = throws)

bool try\_lock\_for (util::steady\_duration const &rel\_time, char const *description, error\_code &ec = throws)

bool try\_lock\_for (util::steady\_duration const &rel\_time, error\_code &ec = throws)

namespace threads

Typedefs

using hpx::threads::thread\_id\_type = typedef thread\_id

using hpx::threads::thread\_self = typedef coroutines::detail::coroutine\_self

Functions

thread\_id\_type get\_self\_id ()

The function get\_self\_id returns the HPX thread id of the current thread (or zero if the current thread is not a HPX thread).

thread\_self *get\_self\_ptr ()

The function get\_self\_ptr returns a pointer to the (OS thread specific) self reference to the current HPX thread.

#include <hpx/synchronization/spinlock\_no\_backoff.hpp>

namespace hpx

namespace lcos

namespace local

struct spinlock\_no\_backoff

#include <spinlock\_no\_backoff.hpp> boost::mutex\_compatible spinlock class

Public Functions

HPX\_NON\_COPYABLE (spinlock\_no\_backoff)

spinlock\_no\_backoff ()

~spinlock\_no\_backoff ()
void lock ()
bool try_lock ()
void unlock ()

Private Members

std::uint64_t v_

#include <hpx/synchronization/counting_semaphore.hpp>
namespace hpx

namespace lcos

namespace local

Typedefs

typedef counting_semaphore_var counting_semaphore

template <typename Mutex = hpx::lcos::local::spinlock, int N = 0>
class counting_semaphore_var

#include <counting_semaphore.hpp> A semaphore is a protected variable (an entity storing a value) or abstract data type (an entity grouping several variables that may or may not be numerical) which constitutes the classic method for restricting access to shared resources, such as shared memory, in a multiprogramming environment. Semaphores exist in many variants, though usually the term refers to a counting semaphore, since a binary semaphore is better known as a mutex. A counting semaphore is a counter for a set of available resources, rather than a locked/unlocked flag of a single resource. It was invented by Edsger Dijkstra. Semaphores are the classic solution to preventing race conditions in the dining philosophers problem, although they do not prevent resource deadlocks.

Counting semaphores can be used for synchronizing multiple threads as well: one thread waiting for several other threads to touch (signal) the semaphore, or several threads waiting for one other thread to touch this semaphore.

Public Functions

counting_semaphore_var (std::int64_t value = N)
Construct a new counting semaphore.

Parameters
• value: [in] The initial value of the internal semaphore lock count. Normally this value should be zero (which is the default), values greater than zero are equivalent to the same number of signals pre-set, and negative values are equivalent to the same number of waits pre-set.

void wait (std::int64_t count = 1)
Wait for the semaphore to be signaled.

Parameters
• count: [in] The value by which the internal lock count will be decremented. At the same time this is the minimum value of the lock count at which the thread is not yielded.
bool try_wait (std::int64_t count = 1)
Try to wait for the semaphore to be signaled.

Return The function returns true if the calling thread was able to acquire the requested amount
of credits. The function returns false if not sufficient credits are available at this point in time.

Parameters
• count: [in] The value by which the internal lock count will be decremented. At the same
time this is the minimum value of the lock count at which the thread is not yielded.

void signal (std::int64_t count = 1)
Signal the semaphore.

std::int64_t signal_all ()

Private Types

typedef Mutex mutex_type

Private Members

mutex_type mtx_

detail::counting_semaphore sem_

#include <hpx/synchronization/recursive_mutex.hpp>

namespace hpx

namespace lcos

namespace local

Typedefs

using hpx::lcos::local::recursive_mutex = typedef detail::recursive_mutex_impl<>

#include <hpx/synchronization/no_mutex.hpp>

namespace hpx

namespace lcos

namespace local

struct no_mutex

Public Functions

void lock ()
bool try_lock ()
void unlock ()
#include <hpx/synchronization/shared_mutex.hpp>

namespace hpx

    namespace lcos

        namespace local

            Typedefs

            typedef detail::shared_mutex shared_mutex

#include <hpx/synchronization/once.hpp>

Defines

HPX_ONCE_INIT

namespace hpx

    namespace lcos

        namespace local

            Functions

            template<typename F, typename... Args>
            void call_once(once_flag &flag, F &&f, Args&&... args)

            struct once_flag

                Public Functions

                HPX_NON_COPYABLE(once_flag)

                once_flag()

                Private Members

                std::atomic<long> status_

                lcos::local::event event_

                Friends

                template<typename F, typename... Args>
                void call_once(once_flag &flag, F &&f, Args&&... args)

#include <hpx/synchronization/latch.hpp>

namespace hpx
namespace lcos

namespace local

class latch
#include <latch.hpp>
Latches are a thread coordination mechanism that allow one or more threads to block until an operation is completed. An individual latch is a single-use object; once the operation has been completed, the latch cannot be reused.

A latch maintains an internal counter_ that is initialized when the latch is created. Threads may block at a synchronization point waiting for counter_ to be decremented to 0. When counter_ reaches 0, all such blocked threads are released.

Calls to countdown_and_wait(), count_down(), wait(), is_ready(), count_up(), and reset() behave as atomic operations.

Note A local::latch is not a LCO in the sense that it has no global id and it can’t be triggered using the action (parcel) mechanism. Use lcos::latch instead if this is required. It is just a low level synchronization primitive allowing to synchronize a given number of threads.

Public Functions

HPX_NON_COPYABLE(latch)
latch (std::ptrdiff_t count)
Initialize the latch
Requires: count >= 0. Synchronization: None Postconditions: counter_ == count.

~latch()
Requires: No threads are blocked at the synchronization point.

Note May be called even if some threads have not yet returned from wait() or count_down_and_wait(), provided that counter_ is 0.

Note The destructor might not return until all threads have exited wait() or count_down_and_wait().

Note It is the caller’s responsibility to ensure that no other thread enters wait() after one thread has called the destructor. This may require additional coordination.

void countdown_and_wait()
Decrement counter_ by 1. Blocks at the synchronization point until counter_ reaches 0.

Requires: counter_ > 0.

Synchronization: Synchronizes with all calls that block on this latch and with all is_ready calls on this latch that return true.

Exceptions
• Nothing.

void count_down (std::ptrdiff_t n)
Decrement counter_ by n. Does not block.

Requires: counter_ >= n and n >= 0.

Synchronization: Synchronizes with all calls that block on this latch and with all is_ready calls on this latch that return true.

Exceptions
bool is_ready() const
    Returns: counter_ == 0. Does not block.

Exceptions
    • Nothing.

void wait() const
    If counter_ is 0, returns immediately. Otherwise, blocks the calling thread at the synchronization point until counter_ reaches 0.

Exceptions
    • Nothing.

void abort_all()

void count_up(std::ptrdiff_t n)
    Increments counter_ by n. Does not block.
    Requires: n >= 0.

Exceptions
    • Nothing.

void reset(std::ptrdiff_t n)
    Reset counter_ to n. Does not block.
    Requires: n >= 0.

Exceptions
    • Nothing.

Protected Attributes

util::cache_line_data<mutex_type> mtx_
util::cache_line_data<local::detail::condition_variable> cond_
std::atomic<std::ptrdiff_t> counter_
bool notified_

Private Types

typedef lcos::local::spinlock mutex_type

#include <hpx/synchronization/channel_mpmc.hpp>

namespace hpx
    namespace lcos
        namespace local
typedefs

using hpx::lcos::local::channel_mpmc = typedef bounded_channel<T, hpx::lcos::local::spinlock>

template <typename T, typename Mutex = util::spinlock>
class bounded_channel

public functions

bounded_channel (std::size_t size)
bounded_channel (bounded_channel &&rhs)
bounded_channel &operator= (bounded_channel &&rhs)
~bounded_channel ()
bool get (T *val = nullptr) const
bool set (T &t)
std::size_t close ()
std::size_t capacity () const

protected functions

std::size_t close (std::unique_lock<mutex_type> &l)

private types

template<>
using mutex_type = Mutex

private functions

bool is_full (std::size_t tail) const
bool is_empty (std::size_t head) const

private members

hpx::util::cache_aligned_data<mutex_type> mtx_
hpx::util::cache_aligned_data<std::size_t> head_
hpx::util::cache_aligned_data<std::size_t> tail_
std::size_t size_
std::unique_ptr<T[]> buffer_
bool closed_
#include <hpx/synchronization/channel_spsc.hpp>

namespace hpx

namespace lcos

namespace local

    template <typename T>
    class channel_spsc

    Public Functions

    channel_spsc (std::size_t size)
    channel_spsc (channel_spsc &&rhs)
    channel_spsc & operator=(channel_spsc &&rhs)
    ~channel_spsc ()
    bool get (T *val = nullptr) const
    bool set (T &&t)
    std::size_t close ()
    std::size_t capacity () const

    Private Functions

    bool is_full (std::size_t tail) const
    bool is_empty (std::size_t head) const

    Private Members

    hpx::util::cache_aligned_data<std::atomic<std::size_t>> head_
    hpx::util::cache_aligned_data<std::atomic<std::size_t>> tail_
    std::size_t size_
    std::unique_ptr<T[]> buffer_
    std::atomic<bool> closed_

#include <hpx/synchronization/barrier.hpp>

namespace hpx

    namespace lcos
namespace local

class barrier
#include <barrier.hpp>
A barrier can be used to synchronize a specific number of threads, blocking all of the entering threads until all of the threads have entered the barrier.

Note A barrier is not a LCO in the sense that it has no global id and it can’t be triggered using the action (parcel) mechanism. It is just a low level synchronization primitive allowing to synchronize a given number of threads.

Public Functions

barrier (std::size_t number_of_threads)
~barrier ()
void wait ()
   The function wait will block the number of entering threads (as given by the constructor parameter number_of_threads), releasing all waiting threads as soon as the last thread entered this function.

void count_up ()
   The function count_up will increase the number of threads to be waited in wait function.

void reset (std::size_t number_of_threads)
   The function reset will reset the number of threads as given by the function parameter number_of_threads. the newer coming threads executing the function wait will be waiting until total_ is equal to barrier_flag. The last thread exiting the wait function will notify the newer threads waiting and the newer threads will get the reset number_of_threads_. The function reset can be executed while previous threads executing waiting after they have been waken up. Thus total_ can not be reset to barrier_flag which will break the comparison condition under the function wait.

Private Types

typedef lcos::local::spinlock mutex_type

Private Members

static_cast<std::size_t>((1 << (CHAR_BIT * sizeof(std::size_t) - 1)]
std::size_t number_of_threads_
std::size_t total_
mutex_type mtx_
local::detail::condition_variable cond_

#include <hpx/synchronization/sliding_semaphore.hpp>

namespace hpx

namespace lcos

namespace local
Typedefs

typedef sliding_semaphore_var sliding_semaphore

template<typename Mutex = hpx::lcos::local::spinlock>
class sliding_semaphore_var

A semaphore is a protected variable (an entity storing a value) or abstract data type (an entity grouping several variables that may or may not be numerical) which constitutes the classic method for restricting access to shared resources, such as shared memory, in a multiprogramming environment. Semaphores exist in many variants, though usually the term refers to a counting semaphore, since a binary semaphore is better known as a mutex. A counting semaphore is a counter for a set of available resources, rather than a locked/unlocked flag of a single resource. It was invented by Edsger Dijkstra. Semaphores are the classic solution to preventing race conditions in the dining philosophers problem, although they do not prevent resource deadlocks.

Sliding semaphores can be used for synchronizing multiple threads as well: one thread waiting for several other threads to touch (signal) the semaphore, or several threads waiting for one other thread to touch this semaphore. The difference to a counting semaphore is that a sliding semaphore will not limit the number of threads which are allowed to proceed, but will make sure that the difference between the (arbitrary) number passed to set and wait does not exceed a given threshold.

Public Functions

sliding_semaphore_var(std::int64_t max_difference, std::int64_t lower_limit = 0)
Construct a new sliding semaphore.

Parameters
- max_difference: [in] The max difference between the upper limit (as set by wait()) and the lower limit (as set by signal()) which is allowed without suspending any thread calling wait().
- lower_limit: [in] The initial lower limit.

void set_max_difference(std::int64_t max_difference, std::int64_t lower_limit = 0)
Set/Change the difference that will cause the semaphore to trigger.

Parameters
- max_difference: [in] The max difference between the upper limit (as set by wait()) and the lower limit (as set by signal()) which is allowed without suspending any thread calling wait().
- lower_limit: [in] The initial lower limit.

void wait(std::int64_t upper_limit)
Wait for the semaphore to be signaled.

Parameters
- upper_limit: [in] The new upper limit. The calling thread will be suspended if the difference between this value and the largest lower_limit which was set by signal() is larger than the max_difference.

bool try_wait(std::int64_t upper_limit = 1)
Try to wait for the semaphore to be signaled.

Return The function returns true if the calling thread would not block if it was calling wait().

Parameters
• **upper_limit**: [in] The new upper limit. The calling thread will be suspended if the difference between this value and the largest lower_limit which was set by `signal()` is larger than the max_difference.

```cpp
void signal (std::int64_t lower_limit)
Signal the semaphore.
```

**Parameters**
- `lower_limit`: [in] The new lower limit. This will update the current lower limit of this semaphore. It will also re-schedule all suspended threads for which their associated upper limit is not larger than the lower limit plus the max_difference.

```cpp
std::int64_t signal_all()
```

### Private Types

```cpp
typedef Mutex mutex_type
```

### Private Members

```cpp
mutex_type mtx_

detail::sliding_semaphore sem_
```

### Enums

```cpp
enum cv_status
Values:
no_timeout
timeout
error
```

### Public Functions

```cpp
void notify_one (error_code &ec = throws)
void notify_all (error_code &ec = throws)
void wait (std::unique_lock<Mutex> &lock, error_code &ec = throws)
```

```cpp
template <class Predicate>
void wait (std::unique_lock<Mutex> &lock, Predicate pred, error_code& = throws)
```
cv_status wait_until (std::unique_lock<mutex> &lock, util::steady_time_point const &abs_time, error_code &ec = throws)

```cpp
template <typename Predicate>
bool wait_until (std::unique_lock<mutex> &lock, util::steady_time_point const &abs_time, Predicate pred, error_code &ec = throws)
```

cv_status wait_for (std::unique_lock<mutex> &lock, util::steady_duration const &rel_time, error_code &ec = throws)

```cpp
template <typename Predicate>
bool wait_for (std::unique_lock<mutex> &lock, util::steady_duration const &rel_time, Predicate pred, error_code &ec = throws)
```

**Private Types**

typedef lcos::local::spinlock mutex_type

**Private Members**

util::cache_line_data<mutex_type> mtx_

util::cache_line_data<detail::condition_variable> cond_

**Public Functions**

void notify_one (error_code &ec = throws)

void notify_all (error_code &ec = throws)

```cpp
template <class Lock>
void wait (Lock &lock, error_code &ec = throws)
```

```cpp
template <class Lock, class Predicate>
void wait (Lock &lock, Predicate pred, error_code &ec = throws)
```

```cpp
template <typename Lock>
cv_status wait_until (Lock &lock, util::steady_time_point const &abs_time, error_code &ec = throws)
```

```cpp
template <typename Lock, typename Predicate>
bool wait_until (Lock &lock, util::steady_time_point const &abs_time, Predicate pred, error_code &ec = throws)
```

```cpp
template <typename Lock>
cv_status wait_for (Lock &lock, util::steady_duration const &rel_time, error_code &ec = throws)
```

```cpp
template <typename Lock, typename Predicate>
bool wait_for (Lock &lock, util::steady_duration const &rel_time, Predicate pred, error_code &ec = throws)
```

**Private Types**

typedef lcos::local::spinlock mutex_type
Private Members

util::cache_line_data<mutex_type> mtx_
util::cache_line_data<detail::condition_variable> cond_

#include <hpx/synchronization/spinlock_pool.hpp>

namespace hpx

namespace lcos

namespace local

template <typename Tag, std::size_t N = HPX_HAVE_SPINLOCK_POOL_NUM>
class spinlock_pool

Public Static Functions

static lcos::local::spinlock &spinlock_for (void const *pv)

Private Static Attributes

util::cache_aligned_data<lcos::local::spinlock> pool_

class scoped_lock

Public Functions

template<>
HPX_NON_COPYABLE (scoped_lock)

template<>
scoped_lock (void const *pv)

template<>
~scoped_lock ()

template<>
void lock ()

template<>
void unlock ()

Private Members

template<>
hpx::lcos::local::spinlock &sp_
#include <hpx/synchronization/channel_mpsc.hpp>

namespace hpx

namespace lcos

namespace local

## Typedefs

using hpx::lcos::local::channel_mpsc = typedef base_channel_mpsc<T, hpx::lcos::local::spinlock>

template <typename T, typename Mutex = util::spinlock>
class base_channel_mpsc

### Public Functions

base_channel_mpsc(std::size_t size)

base_channel_mpsc(base_channel_mpsc &&rhs)

base_channel_mpsc &operator=(base_channel_mpsc &&rhs)

~base_channel_mpsc()

bool get(T *val = nullptr) const

bool set(T &&t)

std::size_t close() const

std::size_t capacity() const

### Private Types

template<>
using mutex_type = Mutex

### Private Functions

bool is_full(std::size_t tail) const

bool is_empty(std::size_t head) const

### Private Members

hpx::util::cache_aligned_data<std::atomic<std::size_t>> head_

hpx::util::cache_aligned_data<tail_data> tail_

std::size_t size_

std::unique_ptr<T[]> buffer_

std::atomic<bool> closed_

struct tail_data

2.9. API reference
Public Members

```cpp
template<>
mutex_type mtx_

template<>
std::atomic<std::size_t> tail_
```

```cpp
#include <hpx/synchronization/spinlock.hpp>

namespace hpx

namespace lcos

namespace local

struct spinlock

Public Functions

```cpp
HPX_NON_COPYABLE(spinlock)

spinlock(char const *const desc = "hpx::lcos::local::spinlock")

~spinlock()

void lock()

bool try_lock()

void unlock()

Private Functions

bool acquire_lock()

void relinquish_lock()

Private Members

std::uint64_t v_
```

#define

```cpp
#include <hpx/preprocessor/expand.hpp>

Defines

HPX_PP_EXPAND(X)

The HPX_PP_EXPAND macro performs a double macro-expansion on its argument. This macro can be used to produce a delayed preprocessor expansion.

Parameters

- X: Token to be expanded twice

622 Chapter 2. What's so special about HPX?
Example:

```cpp
#define MACRO(a, b, c) (a)(b)(c)
#define ARGS() (1, 2, 3)
HPX_PP_EXPAND(MACRO ARGS()) // expands to (1)(2)(3)
```

```cpp
#include <hpx/preprocessor/cat.hpp>
```

**Defines**

**HPX_PP_CAT**(A, B)

Concatenates the tokens A and B into a single token. Evaluates to AB

**Parameters**

- **A**: First token
- **B**: Second token

```cpp
#include <hpx/preprocessor/strip_parens.hpp>
```

**Defines**

**HPX_PP_STRIP_PARENS**(X)

For any symbol X, this macro returns the same symbol from which potential outer parens have been removed. If no outer parens are found, this macro evaluates to X itself without error.

The original implementation of this macro is from Steven Watanbe as shown in http://boost.2283326.n4.nabble.com/preprocessor-removing-parentheses-td2591973.html#a2591976

```cpp
HPX_PP_STRIP_PARENS(no_parens)
HPX_PP_STRIP_PARENS((with_parens))
```

**Example Usage:**

**Parameters**

- **X**: Symbol to strip parens from

This produces the following output

```cpp
no_parens
with_parens
```

```cpp
#include <hpx/preprocessor/nargs.hpp>
```

**Defines**

**HPX_PP_NARGS**(...)  
Expands to the number of arguments passed in

**Example Usage:**

```cpp
HPX_PP_NARGS(hpx, pp, nargs)
HPX_PP_NARGS(hpx, pp)
HPX_PP_NARGS(hpx)
```
Parameters

- \ldots: The variadic number of arguments

Expands to:

<table>
<thead>
<tr>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

#include <hpx/preprocessor/stringize.hpp>

Defines

HPX_PP_STRINGIZE(X)

The `HPX_PP_STRINGIZE` macro stringizes its argument after it has been expanded. The passed argument `X` will expand to "X". Note that the stringizing operator (`) prevents arguments from expanding. This macro circumvents this shortcoming.

Parameters

- `X`: The text to be converted to a string literal

timing

#include <compatibility/hpx/util/scoped_timer.hpp>
#include <compatibility/hpx/util/steady_clock.hpp>
#include <compatibility/hpx/util/high_resolution_timer.hpp>
#include <compatibility/hpx/util/high_resolution_clock.hpp>
#include <hpx/timing/scoped_timer.hpp>

namespace hpx

namespace util

template<typename T>
struct scoped_timer

Public Functions

scoped_timer(T &t, bool enabled = true)
scoped_timer(scoped_timer const&)
scoped_timer(scoped_timer &&rhs)
~scoped_timer()

scoped_timer &operator=(scoped_timer const &rhs)
scoped_timer &operator=(scoped_timer &&rhs)

bool enabled() const
Private Members

`std::uint64_t started_at_
T *t_`

#include <hpx/timing/steady_clock.hpp>

namespace hpx

namespace util

class steady_duration

Public Functions

`steady_duration(value_type const &rel_time)`

`template<typename Rep, typename Period>
steady_duration(std::chrono::duration<Rep, Period> const &rel_time)`

`value_type const &value() const`

`steady_clock::time_point from_now() const`

Private Types

`typedef steady_clock::duration value_type`

Private Members

`value_type _rel_time`

class steady_time_point

Public Functions

`steady_time_point(value_type const &abs_time)`

`template<typename Clock, typename Duration>
steady_time_point(std::chrono::time_point<Clock, Duration> const &abs_time)`

`value_type const &value() const`

Private Types

`typedef steady_clock::time_point value_type`

Private Members

`value_type _abs_time`
```cpp
#include <hpx/timing/high_resolution_timer.hpp>

namespace hpx

namespace util

    class high_resolution_timer

        Public Functions

        high_resolution_timer()
        high_resolution_timer(double t)
        void restart()
        double elapsed() const
        std::int64_t elapsed_microseconds() const
        std::int64_t elapsed_nanoseconds() const
        double elapsed_max() const
        double elapsed_min() const

        Public Static Functions

        static double now()

        Protected Static Functions

        static std::uint64_t take_time_stamp()

        Private Members

        std::uint64_t start_time_

#include <hpx/timing/high_resolution_clock.hpp>

namespace hpx

namespace util

    struct high_resolution_clock

        Public Static Functions

        static std::uint64_t now()
        static std::uint64_t high_resolution_clock::min()
        static std::uint64_t high_resolution_clock::max()
```
logging
#include <compatibility/hpx/util/logging.hpp>
#include <compatibility/hpx/util/logging/format_fwd.hpp>
#include <compatibility/hpx/util/logging/format.hpp>
#include <compatibility/hpx/util/logging/logging.hpp>
#include <compatibility/hpx/util/logging/writer/format_write.hpp>
#include <compatibility/hpx/util/logging/writer/named_write.hpp>
#include <compatibility/hpx/util/logging/format/named_write.hpp>
#include <compatibility/hpx/util/logging/format/named_write_fwd.hpp>
#include <compatibility/hpx/util/logging/format/op_equal.hpp>
#include <compatibility/hpx/util/logging/format/optimize.hpp>
#include <compatibility/hpx/util/logging/format/array.hpp>
#include <compatibility/hpx/util/logging/format/formatter/thread_id.hpp>
#include <compatibility/hpx/util/logging/format/formatter/spacer.hpp>
#include <compatibility/hpx/util/logging/format/formatter/time_strf.hpp>
#include <compatibility/hpx/util/logging/format/formatter/time.hpp>
#include <compatibility/hpx/util/logging/format/formatter/high_precision_time.hpp>
#include <compatibility/hpx/util/logging/format/formatter/named_spacer.hpp>
#include <compatibility/hpx/util/logging/format/formatter/convert_format.hpp>
#include <compatibility/hpx/util/logging/format/formatter/defaults.hpp>
#include <compatibility/hpx/util/logging/format/destination/convert_destination.hpp>
#include <compatibility/hpx/util/logging/format/destination/named.hpp>
#include <compatibility/hpx/util/logging/format/destination/file.hpp>
#include <compatibility/hpx/util/logging/format/destination/defaults.hpp>
#include <hpx/logging.hpp>

Defines
LAGAS_(lvl)
LPT_(lvl)
LTIM_(lvl)
LPROGRESS_
LHPX_(lvl, cat)
LAPP_(lvl)
LDEB_
LTM_(lvl)
LRT_(lvl)
LOSH_(lvl)
LERR_(lvl)
LLCO_(lvl)
LPCS_(lvl)
LAS_(lvl)
LBT_(lvl)
LFATAL_
LAGAS_CONSOLE_(lvl)
LPT_CONSOLE_(lvl)
LTIM_CONSOLE_(lvl)
LHPX_CONSOLE_(lvl)
LAPP_CONSOLE_(lvl)
LDEB_CONSOLE_
LAGAS_ENABLED(lvl)
LPT_ENABLED(lvl)
LTIM_ENABLED(lvl)
LHPX_ENABLED(lvl)
LAPP_ENABLED(lvl)
LDEB_ENABLED

Functions

\texttt{template <\texttt{typename T}>}
\texttt{bootstrap\_logging const \&operator\langle\langle (bootstrap\_logging const \&l, T\&\&)}

Variables

\texttt{constexpr bootstrap\_logging \texttt{lbt\_}}
\texttt{struct bootstrap\_logging}

\hspace{1cm} \textbf{Public Functions}

\hspace{2cm} \texttt{constexpr bootstrap\_logging()}
#include <hpx/logging/format_fwd.hpp>

#include <hpx/logging/format.hpp>

Include this file when you're using formatters and destinations, and you want to define the logger classes, in a source file (using HPX_DEFINE_LOG)

```cpp
namespace hpx {
    namespace util {
        namespace logging {
            namespace format_and_write {

                struct simple {

                    #include <format.hpp>

                    Formats the message, and writes it to destinations.
                    • calls operator() on the formatters, and operator() on the destinations. Ignores clear_format() commands.

                    If you derive from destination::base, this type can be destination::base::raw_param (see below).

                    Parameters
                    • msg_type: The message to pass to the formatter. This is the type that is passed to the formatter objects and to the destination objects. Thus, it needs to be convertible to the argument to be sent to the formatter objects and to the argument to be sent to the destination objects. Usually, it's the argument you pass on to your destination classes.

                    Example:

                    typedef destination::base< const std::string & > dest_base;
                    // in this case : msg_type = std::string = dest_base::raw_param
                    struct write_to_cout : dest_base {
                        void operator() (param msg) const {
                            std::cout << msg ;
                        }
                    };

                    typedef destination::base< const std::string & > dest_base;
                    // in this case : msg_type = cache_string = dest_base::raw_param
                    struct write_to_file : dest_base, ... {
                        void operator() (param msg) const {
                            context() << msg ;
                        }
                    };

                }
            }
        }
    }
}
```

2.9. API reference
Public Functions

```cpp
simple (msg_type &msg)
```

```cpp
template <class formatter_ptr>
void format (const formatter_ptr &fmt)
```

```cpp
template <class destination_ptr>
void write (const destination_ptr &dest)
```

```cpp
void clear_format ()
```

Protected Attributes

```cpp
msg_type &m_msg
```

```cpp
namespace msg_route
```

Specifies the route: how formatting and writing to destinations take place.

Classes in this namespace specify when formatters and destinations are to be called.

```cpp
See msg_route::simple
```

```cpp
template <class formatter_array, class destination_array>
struct formatter_and_destination_array_holder
```

```cpp
#include <format.hpp>
```

Recomended base class for message routers that need access to the underlying formatter and/or destination array.

Protected Functions

```cpp
formatter_and_destination_array_holder (const formatter_array &formats_, const destination_array &destinations_)
```

```cpp
const formatter_array &formats () const
```

```cpp
const destination_array &destinations () const
```

Private Members

```cpp
const formatter_array &m_formats
```

```cpp
const destination_array &m_destinations
```

```cpp
struct simple
```

```cpp
#include <format.hpp>
```

Represents a simple router: first calls all formatters.

• in the order they were added, then all destinations - in the order they were added

Example:

```cpp
typedef logger< format_write > logger_type;
HPX_DEFINE_LOG_FILTER(g_log_filter, filter::no_ts )
HPX_DEFINE_LOG(g_l, logger_type)
#define L_ HPX_LOG_USE_LOG_IF_FILTER(g_l(), g_log_filter() \= is_{ enabled() )

// add formatters: [idx] [time] message [enter]
g_l() \= writer().add_formatter( write_idx() );
```

(continues on next page)
In the above case:
• First, the formatters are called: `write_idx()` is called, then `write_time()`, then `append_newline()`.
• Then, the destinations are called: `write_to_cout()`, and then `write_to_file()`.

Parameters
• `format_base`: The base class for all formatter classes from your application. See `manipulator`.
• `destination_base`: The base class for all destination classes from your application. See `manipulator`.

Public Types

```cpp
typedef format::base::ptr_type formatter_ptr
typedef destination::base::ptr_type destination_ptr
typedef std::vector<formatter_ptr> f_array
typedef std::vector<destination_ptr> d_array
```

Public Functions

```cpp
template <class formatter_array, class destination_array>
void simple(const formatter_array&, const destination_array&)
```

```cpp
void append_formatter (formatter_ptr fmt)
```

```cpp
void del_formatter (formatter_ptr fmt)
```

```cpp
void append_destination(destination_ptr dest)
```

```cpp
void del_destination(destination_ptr dest)
```

```cpp
template <class format_and_write>
void write(msg_type &msg) const
```

Private Members

```cpp
write_info m_to_write
```

```cpp
struct write_info
```

Public Members

```cpp
f_array formats
```
Include this file when you’re using the logging lib, but don’t necessarily want to use formatters and destinations. If you want to use formatters and destinations, then you can include this one instead:

```cpp
#include <hpx/logging/format_fwd.hpp>
```

```cpp
#include <hpx/logging/writer/format_write.hpp>
#include <hpx/logging/writer/named_write.hpp>
```

```cpp
namespace hpx
namespace util
namespace logging
namespace writer
```

```cpp
struct named_write
```

```cpp
#include <named_write.hpp> Composed of a named formatter and a named destinations. Thus, you can specify the formatting and destinations as strings.

```cpp
#include <hpx/logging/format/named_write.hpp>
```

Contains a very easy interface for using formatters and destinations:
- at construction, specify 2 params: the formatter string and the destinations string
Setting the formatters and destinations to write to is extremely simple:

```cpp
// Set the formatters (first param) and destinations (second step) in one step
auto g_l() = writer().write("%time%($hh:$mm.$ss.$mili) [%idx%] |\n", "cout file(out.txt) debug");

// set the formatter(s)
g_l()->writer().format("%time%($hh:$mm.$ss.$mili) [%idx%] |\n");

// set the destination(s)
g_l()->writer().destination("cout file(out.txt) debug");
```

**Public Functions**

```cpp
named_write()
```

```cpp
void format (const std::string &format_str)
```

sets the format string: what should be before, and what after the original message, separated by “|”

Example: “[%idx%] \n” - this writes “[%idx%] ” before the message, and “\n” after the message
If “|” is not present, the whole message is prepended to the message

```cpp
void format (const std::string &format_before_str, const std::string &format_after_str)
```
sets the format strings (what should be before, and what after the original message)

```cpp
void destination (const std::string &destination_str)
```
sets the destinations string - where should logged messages be outputted

```cpp
void write (const std::string &format_str, const std::string &destination_str)
```
Specifies the formats and destinations in one step.

```cpp
const std::string &format () const
```

```cpp
const std::string &destination () const
```

```cpp
void operator() (msg_type &msg) const
```

```cpp
template <class destination>
void replace_destination (const std::string &name, destination d)
```
Replaces a destination from the named destination.

You can use this, for instance, when you want to share a destination between multiple named writers.

```cpp
template <class formatter>
void replace_formatter (const std::string &name, formatter d)
```
Replaces a formatter from the named formatter.

You can use this, for instance, when you want to share a formatter between multiple named writers.

```cpp
template <class formatter>
void add_formatter (formatter fmt)
```

```cpp
template <class destination>
void add_destination (const std::string &name, destination d)
```

### Private Functions

```cpp
template <class manipulator, class parser_type>
void set_and_configure (manipulator &manip, const std::string &name, parser_type parser)
```

```cpp
void init ()
```

### Private Members

```cpp
formatter::named_spacer_t<formatter::do_convert_format::prepend> m_format_before
```

```cpp
formatter::named_spacer_t<formatter::do_convert_format::append> m_format_after
```

```cpp
destination::named m_destination
```

```cpp
format_write m_writer
```

```cpp
std::string m_format_str
```

```cpp
std::string m_format_before_str
```

```cpp
std::string m_format_after_str
```

```cpp
std::string m_destination_str
```

---

2.9. API reference 633
struct parse_destination

**Public Functions**

bool has_manipulator_name() const

std::string get_manipulator_name() const

void clear()

void add(char c)

**Private Members**

std::string m_manipulator

struct parse_formatter

**Public Functions**

bool has_manipulator_name() const

std::string get_manipulator_name() const

void clear()

void add(char c)

**Private Members**

std::string m_manipulator

#include <hpx/logging/format/named_write.hpp>

#include <hpx/logging/format/named_write_fwd.hpp>

#include <hpx/logging/format/op_equal.hpp>

namespace hpx

namespace util

namespace logging

namespace op_equal

Implements operator== for manipulators.

**Functions**

bool operator==(const same_type_op_equal_top &a, const same_type_op_equal_top &b)

template <class type>
struct same_type_op_equal: public hpx::util::logging::op_equal::same_type_op_equal_base
#include <op_equal.hpp> Implements operator==, which compares two objects. If they have the same type, it will compare them using the type’s member operator==.

The only constraint is that operator== must be a member function

Public Functions

virtual bool equals(const same_type_op_equal_top &other) const

struct same_type_op_equal_base: public virtual hpx::util::logging::op_equal::same_type_op_equal
#include <op_equal.hpp> Base class when you want to implement operator== that will compare based on type and member operator==.

See same_type_op_equal

Subclassed by hpx::util::logging::op_equal::same_type_op_equal< type >

struct same_type_op_equal_top
Subclassed by hpx::util::logging::op_equal::same_type_op_equal_base

Public Functions

virtual bool equals(const same_type_op_equal_top&) const = 0

Protected Functions

same_type_op_equal_top()
virtual ~same_type_op_equal_top()
same_type_op_equal_top(const same_type_op_equal_top&)
It keeps all the modified message in one string. Useful if some formatter needs to access the whole string at once.

reserve_prepend() - the size that is reserved for prepending (similar to string::reserve function)
reserve_append() - the size that is reserved for appending (similar to string::reserve function)
Note: as strings are prepended, reserve_prepend() shrinks. Same goes for append.

Public Types

typedef cache_string_one_str self_type

Public Functions

cache_string_one_str (std::size_t reserve_prepend_, std::size_t reserve_append_,
std::size_t grow_size_ = 10)

Parameters
• reserve_prepend: - how many chars to have space to prepend by default
• reserve_append: - how many chars to have space to append by default
• grow_size: - in case we add a string and there's no room for it, with how much should we grow? We'll grow this much in addition to the added string
  – in the needed direction

cache_string_one_str (const std::string &msg, std::size_t reserve_prepend_ = 10,
std::size_t reserve_append_ = 10, std::size_t grow_size_ = 10)

Parameters
• msg: - the message that is originally cached
• reserve_prepend: - how many chars to have space to prepend by default
• reserve_append: - how many chars to have space to append by default
• grow_size: - in case we add a string and there's no room for it, with how much should we grow? We'll grow this much in addition to the added string
  – in the needed direction

cache_string_one_str (cache_string_one_str &&other)

cache_string_one_str ()

void set_string (const std::string &str)

std::size_t reserve_prepend () const

std::size_t reserve_append () const

std::size_t grow_size () const

void reserve_prepend (std::size_t new_size)

void reserve_append (std::size_t new_size)

void grow_size (std::size_t new_size)

void prepend_string (const char *str)

void append_string (const char *str)

void prepend_string (const std::string &str)
  – pre-pends a string (inserts it at the beginning)
void `append_string` (const `std::string` & `str`) appends a string (inserts it at the end)

template <class stream_type>
void `to_stream` (stream_type & `stream`) const writes the current cached contents to a stream

`const std::string & full_string()` const returns the full string

`operator const std::string&()` const

**Private Functions**

void `resize_string` (`std::size_t reserve_prepend`, `std::size_t reserve_append`) const

`bool is_string_set()` const

**Private Members**

`std::size_t m_reserve_prepend`

`std::size_t m_reserve_append`

`std::size_t m_grow_size`

`std::string m_str`

`bool m_full_msg_computed`

`std::string m_full_msg`

**Private Static Functions**

`static std::size_t str_len (const char * `str`)`

`static std::size_t str_len (const wchar_t * `str`)`

```cpp
#include <hpx/logging/format/array.hpp>
namespace hpx
namespace util
namespace logging
namespace array

template <class base_type>
class `ptr_holder`
#include <array.hpp> holds an array of manipulators (formatters or destinations). It owns them, holding them internally as smart pointers. Each function call is locked.

The base_type must implement operator==
When you call get_ptr() or del(), the type you provide, must implement operator==(const type&, const base_type&)

Public Types

typedef base_type value_type
typedef std::unique_ptr<value_type> ptr_type
typedef std::vector<ptr_type> array_type

Public Functions

template <class derived>
base_type *append(derived val)
template <class derived>
base_type *get_ptr(derived val) const
template <class derived>
void del(derived val)
void del(base_type *p)

Private Members

array_type m_array

#include <hpx/logging/format/formatter/thread_id.hpp>

namespace hpx

namespace util

namespace logging

namespace formatter

Typedefs

typedef thread_id_t thread_id
thread_id_t with default values. See thread_id_t

Writes the thread_id to the log.

Parameters

• convert: [optional] In case there needs to be a conversion between std::(w)string and the string that holds your logged message. See convert_format. For instance, you might use a cached_string class (see optimize namespace).

template <class convert = do_convert_format::prepend>
struct thread_id_t : public is_generic
#include <thread_id.hpp> Writes the thread_id to the log.

Parameters
• **convert**: [optional] In case there needs to be a conversion between std::(w)string and the string that holds your logged message. See convert_format. For instance, you might use a cached_string class (see optimize namespace).

### Public Types

typedef convert convert_type

### Public Functions

```cpp
void operator() (msg_type &msg) const

bool operator==(const thread_id_t &)
```

```cpp
#include <hpx/logging/format/formatter/spacer.hpp>

namespace hpx

namespace util

namespace logging

namespace formatter

### Functions

```cpp
template <class original_formatter>
detail::find_space<original_formatter>::type spacer(const original_formatter &fmt, 
                                                  const char *format_str)
```

Prepends some info, and appends some info to an existing formatter.

The syntax is simple: construct a spacer by passing the original formatter, and the text to space (prepend and append). Use:

- `%` to mean the original formatter text
- anything before "%" is prepended before
- anything after "%" is appended after

Examples:

```cpp
// prefix "[" before index, and append "]" after it.
formatter::spacer( formatter::idx(), "[%] " );

// prefix "{T" before thread_id, and append "}" after it
formatter::spacer( formatter::thread_id(), "{T%} " );
```

When adding a spacer formatter, you’ll do something similar to:

```cpp
g_l() -> writer().add_formatter( formatter::spacer( formatter::idx(), 
                                             "[%] " ) );
```

However, to make this even simpler, I allow an ever easier syntax:

```cpp
// equivalent to the above

// equivalent to the above

g_l() -> writer().add_formatter( formatter::idx(), "[%] " );
```


```cpp
template <class convert, class original_formatter, bool is_generic_formatter>
struct spacer_t : public original_formatter
#

Prepends some info, and appends some info to an existing formatter.

The syntax is simple: construct a spacer by passing the original formatter, and the text to space
(prepend and append). Use:
• % to mean the original formatter text
• anything before "%" is prepended before
• anything after "%" is appended after

Examples:

// prefix "[" before index, and append "]" after it.
formatter::spacer( formatter::idx(), "[%"]

// prefix "{T" before thread_id, and append "}" after it
formatter::spacer( formatter::thread_id(), "{T%}"

When adding a spacer formatter, you’ll do something similar to:

```cpp
g_l() -> writer().add_formatter( formatter::spacer( formatter::idx(), "%" );
```  

However, to make this even simpler, I allow an even easier syntax:

```cpp
// equivalent to the above
```  

```cpp
g_l() -> writer().add_formatter( formatter::idx(), "[%" );
```  

Public Types

typedef original_formatter::param param

typedef original_formatter spacer_base

Public Functions

```cpp
struct_t (const original_formatter &fmt, const char *format_str)

void operator () (param msg) const
```  

Private Functions

```cpp
void parse_format (const std::string &format_str)
```  

Private Members

```cpp
std::string m_prefix
std::string m_suffix
```  

```cpp
template <class convert, class original_formatter>
```  

```cpp
struct spacer_t<convert, original_formatter, true> : public original_formatter
```  

Public Types

typedef original_formatter spacer_base

Chapter 2. What’s so special about HPX?
Public Functions

\texttt{spacer_t} (const original_formatter &fmt, const char *\textit{format_str})

void \texttt{operator()} (msg_type &msg) const

Private Functions

void \texttt{parse_format} (const std::string &\textit{format_str})

Private Members

\texttt{std::string} \texttt{m\_prefix}
\texttt{std::string} \texttt{m\_suffix}

#include <hpx/logging/format/formatter/time_strf.hpp>

namespace hpx

namespace util

namespace logging

namespace formatter

Typedefs

typedef time_strf_t \texttt{time\_strf}

time_strf_t with default values. See time_strf_t

Prefixes the message with the time, by using strftime function. You pass the format string at construction.

Parameters

\begin{itemize}
\item \texttt{msg\_type}: The type that holds your logged message.
\item \texttt{convert}: [optional] In case there needs to be a conversion between std::(w)string and the string that holds your logged message. See convert_format. For instance, you might use a cached_string class (see optimize namespace).
\end{itemize}

template <class convert = do_convert_format::prepend>
struct time_strf_t : public is_generic

#include <time_strf.hpp> Prefixes the message with the time, by using strftime function. You pass the format string at construction.

Parameters

\begin{itemize}
\item \texttt{msg\_type}: The type that holds your logged message.
\item \texttt{convert}: [optional] In case there needs to be a conversion between std::(w)string and the string that holds your logged message. See convert_format. For instance, you might use a cached_string class (see optimize namespace).
\end{itemize}

Public Types

typedef convert \texttt{convert\_type}
Public Functions

time_strf_t (const std::string &format, bool localtime)
constructs a time_strf object

Parameters
• format: the time format, strftime-like
• localtime: if true, use localtime, otherwise global time

void operator() (msg_type &msg) const
bool operator==(const time_strf_t &other) const

Private Members

std::string m_format
bool m_localtime

#include <hpx/logging/format/formatter/time.hpp>
namespace hpx
namespace util
namespace logging
namespace formatter

Typedefs

typedef time_t time

time_t with default values. See time_t

Prefixes the message with the time. You pass the format string at construction. It’s friendlier than write_time_strf (which uses strftime).

The format can contain escape sequences: $dd - day, 2 digits $MM - month, 2 digits $yy - year, 2 digits $yyyy - year, 4 digits $hh - hour, 2 digits $mm - minute, 2 digits $ss - second, 2 digits
Example: time("Today is $dd/$MM/$yyyy");
Note: for a high precision clock, try high_precision_time (uses hpx::util::date_time)

Parameters
• convert: [optional] In case there needs to be a conversion between std::(w)string and
the string that holds your logged message. See convert_format. For instance, you might
use a cached_string class (see optimize namespace).

template <class convert = do_convert_format::prepend>
struct time_t: public is_generic, public non_const_context<hpx::util::logging::detail::time_format_holder>
#include <time.hpp> Prefixes the message with the time. You pass the format string at con-
struction.
It’s friendlier than write_time_strf (which uses strftime).
The format can contain escape sequences: $dd - day, 2 digits $MM - month, 2 digits $yy - year, 2 digits $yyyy - year, 4 digits $hh - hour, 2 digits $mm - minute, 2 digits $ss - second, 2 digits
Example: time(“Today is $dd/$MM/$yyyy”);
Note: for a high precision clock, try high_precision_time (uses hpx::util::date_time)

Parameters
- **convert**: [optional] In case there needs to be a conversion between std::(w)string and the string that holds your logged message. See convert_format. For instance, you might use a cached_string class (see optimize namespace).

Public Types

```cpp
typedef convert convert_type
```

```cpp
typedef non_const_context<hpx::util::logging::detail::time_format_holder> non_const_context_base
```

Public Functions

```cpp
time_t(const std::string &format)
```
- constructs a time object

```cpp
void write_time(msg_type &msg, time_t val) const
```

```cpp
void operator()(msg_type &msg) const
```

```cpp
bool operator==(const time_t &other) const
```

```cpp
void configure(const std::string &str)
```
- configure through script
- the string = the time format

```cpp
#include <hpx/logging/format/formatter/high_precision_time.hpp>
```

namespace hpx

```cpp
namespace util
```

```cpp
namespace logging
```

```cpp
namespace formatter
```

**Typedefs**

```cpp
typedef high_precision_time_t high_precision_time
```
- high_precision_time_t with default values. See high_precision_time_t
- Prefixes the message with a high-precision time . You pass the format string at construction.

```cpp
#include <hpx/logging/format/formatter/high_precision_time.hpp>
```

Internally, it uses hpx::util::date_time::microsec_time_clock. So, our precision matches this class.
The format can contain escape sequences: $dd - day, 2 digits $MM - month, 2 digits $yy - year, 2 digits $yyyy - year, 4 digits $hh - hour, 2 digits $mm - minute, 2 digits $ss - second, 2 digits $mili - milliseconds $micro - microseconds (if the high precision clock allows; otherwise, it pads zeros) $nano - nanoseconds (if the high precision clock allows; otherwise, it pads zeros)

Example:

```cpp
daytime::high_precision_time("$mm:$ss:$micro");
```

**Parameters**

- `convert`: [optional] In case there needs to be a conversion between std::(w)string and the string that holds your logged message. See `convert_format`. For instance, you might use a `cached_string` class (see `optimize` namespace).

```cpp
template <class convert = do_convert_format::prepend>
class high_precision_time_t : public is_generic, public non_const_context<hpx::util::logging::detail::time_format_holder>
#include <high_precision_time.hpp>
Prefixes the message with a high-precision time. You pass the format string at construction.
```

Internally, it uses `hpx::util::date_time::microsec_time_clock`. So, our precision matches this class.

The format can contain escape sequences: $dd - day, 2 digits $MM - month, 2 digits $yy - year, 2 digits $yyyy - year, 4 digits $hh - hour, 2 digits $mm - minute, 2 digits $ss - second, 2 digits $mili - milliseconds $micro - microseconds (if the high precision clock allows; otherwise, it pads zeros) $nano - nanoseconds (if the high precision clock allows; otherwise, it pads zeros)

Example:

```cpp
daytime::high_precision_time("$mm:$ss:$micro");
```

**Parameters**

- `convert`: [optional] In case there needs to be a conversion between std::(w)string and the string that holds your logged message. See `convert_format`. For instance, you might use a `cached_string` class (see `optimize` namespace).

**Public Types**

```cpp
typedef convert convert_type
```

```cpp
typedef non_const_context<hpx::util::logging::detail::time_format_holder> non_const_context_base
```

**Public Functions**

```cpp
class high_precision_time_t (const std::string &format)
constructs a high_precision_time object
```

```cpp
void write_high_precision_time (msg_type &msg,
std::chrono::time_point<std::chrono::system_clock> val) const
```

```cpp
void operator() (msg_type &msg) const
```

```cpp
bool operator==(const high_precision_time_t &other) const
```
void configure(const std::string &str)
  configure through script
  the string = the time format

#include <hpx/logging/format/formatter/named_spacer.hpp>

namespace hpx

namespace util

namespace logging

namespace formatter

template <class convert>
struct named_spacer_t : public is_generic, public non_const_context<detail::named_spacer_context<convert> >
#include <named_spacer.hpp> Allows you to contain multiple formatters, and specify a spacer between them. You have a spacer string, and within it, you can escape your contained formatters.

#include <hpx/logging/format/formatter/named_spacer.hpp>

This allows you:
  • to hold multiple formatters
  • each formatter is given a name, when being added
  • you have a spacer string, which contains what is to be prepended or appended to the string (by default, prepended)
  • a formatter is escaped with '% chars, like this "name"
  • if you want to write the '%, just double it, like this: "this %% gets written"

Example:

#define L_ HPX_LOG_USE_LOG_IF_FILTER(g_l(), g_log_filter()->is_enabled())

g_l()->writer().add_formatter(formatter::named_spacer("[%index%]
  %time% (T%thread%) ")
  .add("index", formatter::idx())
  .add("thread", formatter::thread_id())
  .add("time", formatter::time("$mm") );

Assuming you’d use the above in code

int i = 1;
L_ << "this is so cool " << i++;
L_ << "this is so cool again " << i++;

You could have an output like this:

[1] 53 (T3536) this is so cool 1
[2] 54 (T3536) this is so cool again 2
Public Types

typedef non_const_context<detail::named_spacer_context<convert>> context_base

Public Functions

named_spacer_t (const std::string &str = std::string())

named_spacer_t &string (const std::string &str)

template <class formatter>
named_spacer_t &add (const std::string &name, formatter fmt)

void del (const std::string &name)

void configure_inner (const std::string &name, const std::string &configure_str)

void operator() (msg_type &msg) const

bool operator== (const named_spacer_t &other) const

#include <hpx/logging/format/formatter/convert_format.hpp>

namespace hpx

    namespace util

    namespace logging

    namespace formatter

    struct do_convert_format

    struct append

    Public Static Functions

    template <class string>
    static const std::string &get_underlying_string (const string &str)

    template <class string>
    static void write (const char *src, string &dest)

    template <class src_type, class string>
    static void write (const src_type &src, string &dest)

    template <class src_type, class string>
    static void write (src_type &src, string &dest)

    struct prepend
Public Static Functions

template <class string>
static const std::string &get_underlying_string(const string &str)

template <class string>
static void write(const char *src, string &dest)

template <class src_type, class string>
static void write(const src_type &src, string &dest)

template <class src_type, class string>
static void write(src_type &src, string &dest)

namespace convert

Allows format conversions.

• In case you’re using a formatter that does not match your string type

In case you want to use a formatter developed by someone else (for instance, a formatter pro-
vided by this lib), perhaps you’re using another type of string to hold the message

• thus, you need to provide a conversion function

Example: FIXME

> convert_format::prepend

explain that you can extend the following - since they’re namespaces!!! so that you can “inject”
your own write function in the convert_format::prepend/orwhatever namespace, and then it’ll
be automatically used!

namespace append

Functions

void write(const std::string &src, std::string &dest)

void write(const std::string &src, hpx::util::logging::optimize::cache_string_one_str &dest)

void write(const char *src, std::string &dest)

void write(const char *src, hpx::util::logging::optimize::cache_string_one_str &dest)

namespace prepend

Example: write_time

Functions

void write(const char *src, std::string &dest)

void write(const std::string &src, std::string &dest)

void write(const std::string &src, hpx::util::logging::optimize::cache_string_one_str &dest)

void write(const char *src, hpx::util::logging::optimize::cache_string_one_str &dest)
#include <hpx/logging/format/formatter/defaults.hpp>

namespace hpx

namespace util

namespace logging

namespace formatter

Typedefs

typedef idx_t idx

idx_t with default values. See idx_t

prefixes each message with an index. Example:

```
L_ << "my message";
L_ << "my 2nd message";
```

This will output something similar to:

```
[1] my message
[2] my 2nd message
```

Parameters

• convert: [optional] In case there needs to be a conversion between std::(w)string and the string that holds your logged message. See convert_format. For instance, you might use a cached_string class (see optimize namespace).

typedef append_newline_t append_newline

append_newline_t with default values. See append_newline_t

Appends a new line.

Parameters

• convert: [optional] In case there needs to be a conversion between std::(w)string and the string that holds your logged message. See convert_format. For instance, you might use a cached_string class (see optimize namespace).

typedef append_newline_if_needed_t append_newline_if_needed

append_newline_if_needed_t with default values. See append_newline_if_needed_t

Appends a new line, if not already there.

Parameters

• convert: [optional] In case there needs to be a conversion between std::(w)string and the string that holds your logged message. See convert_format. For instance, you might use a cached_string class (see optimize namespace).

```template <class convert = do_convert_format::append>
struct append_newline_if_needed_t : public is_generic

#include <defaults.hpp> Append a new line, if not already there.

Parameters

• convert: [optional] In case there needs to be a conversion between std::(w)string and the string that holds your logged message. See convert_format. For instance, you might use a cached_string class (see optimize namespace).```
Public Types

typedef convert convert_type

Public Functions

void operator() (msg_type &str) const

bool operator==(const append_newline_if_needed_t&) const

template <class convert = do_convert_format::append>
struct append_newline_t : public is_generic
#include <defaults.hpp> Appends a new line.

Parameters
• convert: [optional] In case there needs to be a conversion between std::(w)string and the string that holds your logged message. See convert_format. For instance, you might use a cached_string class (see optimize namespace).

Public Types

typedef convert convert_type

Public Functions

void operator() (msg_type &str) const

bool operator==(const append_newline_t&) const

template <class convert = do_convert_format::prepend>
struct idx_t : public is_generic, public formatter::non_const_context<std::uint64_t>
#include <defaults.hpp> prefixes each message with an index.

Example:

L_ << "my message";
L_ << "my 2nd message";

This will output something similar to:

[1] my message
[2] my 2nd message

Parameters
• convert: [optional] In case there needs to be a conversion between std::(w)string and the string that holds your logged message. See convert_format. For instance, you might use a cached_string class (see optimize namespace).
Public Functions

idx_t()

void operator() (msg_type & str) const

bool operator== (const idx_t & ) const

#include <hpx/logging/format/destination/convert_destination.hpp>

namespace hpx

namespace util

namespace logging

namespace destination

namespace convert

Allows writing messages to destinations.

It has 2 function overloads:

- write(message, output) - writes the given message, to the given output
- do_convert(message, into<other_type>());

FIXME

Functions

template <class obj>
void write (const obj & m, std::ostream & out)

void write (const char * m, std::ostream & out)

const char * do_convert (const char * c, const into<const char *>& )

const char * do_convert (const std::string & s, const into<const char *>& )

const std::string & do_convert (const std::string & s, const into<std::string>& )

#include <hpx/logging/format/destination/named.hpp>

namespace hpx

namespace util

namespace logging

namespace destination
struct named : public is_generic, public non_const_context<detail::named_context>
#include <named.hpp> Allows you to contain multiple destinations, give each such destination a name. Then, at run-time, you can specify a format string which will specify which destinations to be called, and on what order.

This allows you:
• to hold multiple destinations
• each destination is given a name, when being added. The name must not contain spaces and must not start with ‘+’/’-’ signs
• you have a format string, which contains what destinations to be called, and on which order
The format string contains destination names, separated by space.

When a message is written to this destination, I parse the format string. When a name is encountered, if there’s a destination corresponding to this name, I will call it.

Example:

```cpp
g_l()->writer().add_destination(
    destination::named("cout out debug")
    .add( "cout", destination::cout())
    .add( "debug", destination::dbg_window() )
    .add( "out", destination::file("out.txt"))
);
```

In the above code, we’ll write to 3 destinations, in the following order:
• first, to the console
• second, to the out.txt file
• third, to the debug window

Public Types

typedef non_const_context<detail::named_context> non_const_context_base

Public Functions

named (const std::string &format_string = std::string()) constructs the named destination

Parameters
• named_name: name of the named
• set: [optional] named settings - see named_settings class, and dealing_with_flags

void operator() (const msg_type &msg) const

template <class destination>
name &string (const std::string &str)

named &add (const std::string &name, destination dest)

del (const std::string &name)

configure_inner (const std::string &name, const std::string &configure_str)

operator== (const named &other) const
#include <hpx/logging/format/destination/file.hpp>

namespace hpx

    namespace util

        namespace logging

            namespace destination

                struct file : public is_generic, public non_const_context<detail::file_info>

                    #include <file.hpp>  Writes the string to a file.

                Public Types

                    typedef non_const_context<detail::file_info> non_const_context_base
                    typedef boost::detail::spinlock mutex_type

                Public Functions

                    file (const std::string &file_name, file_settings set = file_settings())
                        constructs the file destination

                        Parameters
                        • file_name: name of the file
                        • set: [optional] file settings - see file_settings class, and dealing_with_flags

                    void operator() (const msg_type &msg) const
                    bool operator==(const file &other) const
                    void configure (const std::string &str)
                        configure through script right now, you can only specify the file name

                Public Static Attributes

                    mutex_type mtx_

                struct file_settings

                    #include <file.hpp>  settings for when constructing a file class. To see how it’s used, see dealing_with_flags.

                Public Types

                    typedef hpx::util::logging::detail::flag<file_settings> flag

                Public Functions

                    file_settings()
Public Members

*flag::t<bool> flush_each_time*  
if true (default), flushes after each write

*flag::t<bool> initial_overwrite*  

*flag::t<bool> do_append*  

*flag::t<std::ios_base::openmode> extra_flags*  
just in case you have some extra flags to pass, when opening the file

```cpp
#include <hpx/logging/format/destination/defaults.hpp>

namespace hpx

namespace util

namespace logging

namespace destination

struct cerr : public is_generic
#include <defaults.hpp>  
Writes the string to cerr.

Public Functions

void operator() (const msg_type &msg) const

bool operator==(const cerr&) const

struct cout : public is_generic
#include <defaults.hpp>  
Writes the string to console.

Public Functions

void operator() (const msg_type &msg) const

bool operator==(const cout&) const

struct dbg_window : public is_generic
#include <defaults.hpp>  
Writes the string to output debug window.
For non-Windows systems, this is the console.

Public Functions

void operator() (const msg_type &msg) const

bool operator==(const dbg_window&) const

struct stream : public is_generic, public non_const_context<std::ostream*>  
#include <defaults.hpp>  
writes to stream.

Note: The stream must outlive this object! Or, clear() the stream, before the stream is deleted.
Public Types

typedef std::ostream stream_type

typedef non_const_context<stream_type *> non_const_context_base

Public Functions

stream (stream_type *s)

stream (stream_type &s)

void operator() (const msg_type &msg) const

bool operator== (const stream &other) const

void set_stream (stream_type *p)

resets the stream. Further output will be written to this stream

void clear ()

clears the stream. Further output will be ignored

local_lcos

#include <compatibility/hpx/lcos/local/packaged_task.hpp>

#include <compatibility/hpx/lcos/local/promise.hpp>

#include <compatibility/hpx/lcos/local/trigger.hpp>

#include <compatibility/hpx/lcos/local/composable_guard.hpp>

#include <compatibility/hpx/lcos/local/receive_buffer.hpp>

#include <compatibility/hpx/lcos/local/channel.hpp>

#include <compatibility/hpx/lcos/local/and_gate.hpp>

#include <compatibility/hpx/lcos/local/conditional_trigger.hpp>

#include <hpx/local_lcos/packaged_task.hpp>

namespace hpx

namespace lcos

namespace local


template <typename R, typename... Ts>
template<class packaged_task<R (Ts...) >

Public Functions

packaged_task ()
template <typename F, typename FD = typename std::decay<F>::type, typename Enable = typename std::enable_if<!std::is_same<FD, packaged_task>::value && traits::is_invocable_r<R, FD&, Ts...>::value>::type> packaged_task (F && f)

template <typename Allocator, typename F, typename FD = typename std::decay<F>::type, typename Enable = typename std::enable_if<!std::is_same<FD, packaged_task>::value && traits::is_invocable_r<R, FD&, Ts...>::value>::type> packaged_task (std::allocator_arg_t, Allocator && a, F && f)

packaged_task (packaged_task && rhs)

packaged_task & operator= (packaged_task && rhs)

void swap (packaged_task & rhs)

void operator() (Ts... vs)

lcos::future<R> get_future (error_code & ec = throws)

bool valid () const

void reset (error_code & ec = throws)

void set_exception (std::exception_ptr const & e)

Private Types

typedef util::unique_function_nonser<R (Ts...) > function_type

Private Functions

template <typename... Vs>
void invoke_impl (std::false_type, Vs&&... vs)

template <typename... Vs>
void invoke_impl (std::true_type, Vs&&... vs)

Private Members

function_type function_

local::promise<R> promise_

#include <hpx/local_lcos/promise.hpp>

namespace hpx

namespace lcos

namespace local

Functions

template <typename R>
void swap (promise<R> & x, promise<R> & y)

template <typename R>
class promise : public hpx::lcos::local::detail::promise_base<R>
Public Functions

```
promise()

template <typename Allocator>
promise(std::allocator_arg_t, Allocator const &a)

promise(promise &&other)

~promise()

promise &operator=(promise &&other)

void swap(promise &other)

bool valid() const

void setValue(R const &r)

void setValue(R &&r)

template <typename... Ts>
void setValue(Ts&&... ts)

void set_exception(std::exception_ptr e)
```

Private Types

```cpp
typedef detail::promise_base<R> base_type

template <typename R>
template<>

class promise<R&> : public hpx::lcos::local::detail::promise_base<R&>
```

Public Functions

```
promise()

template <typename Allocator>
promise(std::allocator_arg_t, Allocator const &a)

promise(promise &&other)

~promise()

promise &operator=(promise &&other)

void swap(promise &other)

bool valid() const

void setValue(R &r)

void set_exception(std::exception_ptr e)
```
Private Types

typedef detail::promise_base<R&> base_type

template <>
template<>
class promise<void> : public hpx::lcos::local::detail::promise_base<void>

Public Functions

promise()

template <typename Allocator>
promise(std::allocator_arg_t, Allocator const &a)
promise(promise &&other)

~promise()
promise &operator=(promise &&other)

void swap(promise &other)

bool valid() const

void set_value()

void set_exception(std::exception_ptr e)

Private Types

typedef detail::promise_base<void> base_type

#include <hpx/local_lcos/trigger.hpp>

namespace hpx

namespace lcos

namespace local

template <typename Mutex = local::spinlock>
struct base_trigger

Public Functions

base_trigger()

base_trigger(base_trigger &&rhs)

base_trigger &operator=(base_trigger &&rhs)

future<void> get_future(std::size_t *generation_value = nullptr, error_code &ec = hpx::throws)

get a future allowing to wait for the trigger to fire
bool set(error_code &ec = throws)
    Trigger this object.

void synchronize(std::size_t generation_value, char const *function_name = "base_and_gate<>::synchronize", error_code &ec = throws)
    Wait for the generational counter to reach the requested stage.

std::size_t next_generation()

std::size_t generation() const

Protected Types
typedef Mutex mutex_type

Protected Functions
bool trigger_conditions(error_code &ec = throws)

template<typename Lock>
void synchronize(std::size_t generation_value, Lock &l, char const *function_name = "base_and_gate<>::synchronize", error_code &ec = throws)

Private Types
typedef std::list<conditional_trigger *> condition_list_type

Private Functions
bool test_condition(std::size_t generation_value)

Private Members
mutex_type mtx_
lcos::local::promise<void> promise_
std::size_t generation_
condition_list_type conditions_

struct manage_condition

Public Functions
template<>
manage_condition(base_trigger &gate, conditional_trigger &cond)

template<>
~manage_condition()

template<typename Condition>
template<>
future<void> get_future(Condition &&func, error_code &ec = hpx::throws)
Public Members

```cpp
template<>
base_trigger &this_

template<>
condition_list_type::iterator it_
```

struct trigger: public hpx::lcos::local::base_trigger<no_mutex>

Public Functions

```cpp
trigger()

trigger(trigger &&rhs)

trigger &operator=(trigger &&rhs)

template <typename Lock>
void synchronize(std::size_t generation_value, Lock &l, char const *function_name = "trigger::synchronize", error_code &ec = throws)
```

Private Types

```cpp
typedef base_trigger<no_mutex> base_type
```

#include <hpx/local_lcos/composable_guard.hpp>

namespace hpx

namespace lcos

namespace local

Functions

void run_guarded(guard &guard, detail::guard_function task)

Conceptually, a guard acts like a mutex on an asynchronous task. The mutex is locked before the task runs, and unlocked afterwards.

```cpp
template <typename F, typename... Args>
void run_guarded(guard &guard, F &&f, Args&&... args)
```

void run_guarded(guard_set &guards, detail::guard_function task)

Conceptually, a guard_set acts like a set of mutexes on an asynchronous task. The mutexes are locked before the task runs, and unlocked afterwards.

```cpp
template <typename F, typename... Args>
void run_guarded(guard_set &guards, F &&f, Args&&... args)
```

class guard: public hpx::lcos::local::detail::debug_object
Public Functions

`guard()`

`~guard()`

Public Members
detail::guard_atomic task
class guard_set : public hpx::lcos::local::detail::debug_object

Public Functions

`guard_set()`

`~guard_set()`

`std::shared_ptr<guard> get (std::size_t i)`

`void add (std::shared_ptr<guard> const &guard_ptr)`

`std::size_t size()`

Private Functions

`void sort()`

Private Members

`std::vector<std::shared_ptr<guard>> guards`

`bool sorted`

Friends

`void run_guarded (guard_set &guards, detail::guard_function task)`

Conceptually, a `guard_set` acts like a set of mutexes on an asynchronous task. The mutexes are locked before the task runs, and unlocked afterwards.

```
#include <hpx/local_lcos/receive_buffer.hpp>

namespace hpx

namespace lcos

namespace local

template <typename T, typename Mutex = lcos::local::spinlock>
struct receive_buffer
```
Public Functions

receive_buffer()

receive_buffer (receive_buffer &other)

~receive_buffer()

receive_buffer &operator=(receive_buffer &other)

hpx::future<T> receive (std::size_t step)

bool try_receive (std::size_t step, hpx::future<T> *f = nullptr)

template <typename Lock = hpx::lcos::local::no_mutex>
void store_received (std::size_t step, T &val, Lock *lock = nullptr)

bool empty() const

std::size_t cancel_waiting (std::exception_ptr const &e, bool force_delete_entries = false)

Protected Types

typedef Mutex mutex_type

typedef hpx::lcos::local::promise<T> buffer_promise_type

typedef std::map<std::size_t, std::shared_ptr<entry_data>> buffer_map_type

typedef buffer_map_type::iterator iterator

Protected Functions

iterator get_buffer_entry (std::size_t step)

Private Members

mutex_type mtx_

buffer_map_type buffer_map_

struct entry_data

Public Functions

template<>
HPX_NON_COPYABLE (entry_data)

template<>
entry_data ()

template<>
hpx::future<T> get_future ()

template <typename Val>
template<>
void set_value (Val &&val)

template<>
bool cancel (std::exception_ptr const &e)

**Public Members**

template<>
buffer.promise_type promise_

template<>
bool can_be_deleted_

template<>
bool value_set_

**struct erase_on_exit**

**Public Functions**

template<>
erase_on_exit (buffer.map_type &buffer_map, iterator it)

template<>
~erase_on_exit ()

**Public Members**

template<>
buffer.map_type &buffer_map_

template<>
iterator it_

**template<typename Mutex>**
template<>
struct receive_buffer<void, Mutex>

template<>
struct receive_buffer<void, Mutex>

**Public Functions**

receive_buffer ()

receive_buffer (receive_buffer &&other)

~receive_buffer ()

receive_buffer &&operator= (receive_buffer &&other)

hpx::future<void> receive (std::size_t step)

bool try_receive (std::size_t step, hpx::future<void> *f = nullptr)

**template<typename Lock = hpx::lcos::local::no_mutex>**
void store_received (std::size_t step, Lock *lock = nullptr)

bool empty () const

std::size_t cancel_waiting (std::exception_ptr const &e, bool force_delete_entries = false)
Protected Types

typedef Mutex mutex_type

typedef hpx::lcos::local::promise< void > buffer_promise_type

typedef std::map< std::size_t, std::shared_ptr< entry_data > > buffer_map_type

typedef buffer_map_type::iterator iterator

Protected Functions

iterator get_buffer_entry ( std::size_t step )

Private Members

mutex_type mtx_

buffer_map_type buffer_map_

template<> struct entry_data

Public Functions

template<> HPX_NON_COPYABLE ( entry_data )

template<> entry_data ()

template<> hpx::future< void > get_future ()

template<> void set_value ()

template<> bool cancel ( std::exception_ptr const & e )

Public Members

template<> buffer_promise_type promise_

template<> bool can_be_deleted_

template<> bool value_set_

template<> struct erase_on_exit
Public Functions

template<>
erase_on_exit (buffer_map_type &buffer_map, iterator it)

template<>
~erase_on_exit ()

Public Members

template<>
buffer_map_type &buffer_map_

template <>
iterator it_

#include <hpx/local_lcos/channel.hpp>

namespace hpx

namespace lcos

namespace local

template <typename T>
class channel : protected hpx::lcos::local::detail::channel_base<T>

Public Types

typedef T value_type

Public Functions

channel ()

Private Types

typedef detail::channel_base<T> base_type

Friends

friend hpx::lcos::local::channel_iterator<T> 
friend hpx::lcos::local::receive_channel<T> 
friend hpx::lcos::local::send_channel<T>

template <>
template<>
class channel<void> : protected hpx::lcos::local::detail::channel_base<void>

Public Types

typedef void value_type
Public Functions

channel()

Private Types

typedef detail::channel_base<void> base_type

Friends

friend hpx::lcos::local::channel_iterator<void>
friend hpx::lcos::local::receive_channel<void>
friend hpx::lcos::local::send_channel<void>

```
template<typename T>
class channel_async_iterator: public hpx::util::iterator_facade<channel_async_iterator<T>, hpx::future<T>, std::input_iterator_tag, hpx::future<T>>
```

Public Functions

channel_async_iterator()

channel_async_iterator (detail::channel_base<T> const *c)

Private Types

typedef hpx::util::iterator_facade<channel_async_iterator<T>, hpx::future<T>, std::input_iterator_tag, hpx::future<T>>

Private Functions

```std::pair<hpx::future<T>, bool> get_checked() const
bool equal (channel_async_iterator const &rhs) const
void increment()```

```
base_type::reference dereference() const
```

Private Members

```
hpx::intrusive_ptr<detail::channel_impl_base<T>> channel_
```

```
std::pair<hpx::future<T>, bool> data_
```

Friends

```
frend hpx::lcos::local::hpx::util::iterator_core_access
```

```
template<typename T>
class channel_iterator: public hpx::util::iterator_facade<channel_iterator<T>, T const, std::input_iterator_tag, hpx::future<T>, T const, std::input_iterator_tag, hpx::future<T>>
```

2.9. API reference 665
Public Functions

channel_iterator()
channel_iterator(detail::channel_base<T> const *c)
channel_iterator(receive_channel<T> const *c)

Private Types

typedef hpx::util::iterator_facade<channel_iterator<T>, T const, std::input_iterator_tag> base_type

Private Functions

std::pair<T, bool> get_checked() const
bool equal(channel_iterator const &rhs) const
void increment() const

base_type::reference dereference() const

Private Members

hpx::intrusive_ptr<detail::channel_impl_base<T>> channel_
std::pair<T, bool> data_

Friends

friend hpx::lcos::local::hpx::util::iterator_core_access

template <>
template<>
class channel_iterator<void>: public hpx::util::iterator_facade<channel_iterator<void>, util::unused_type const, std::input_iterator_tag>

Public Functions

channel_iterator()
channel_iterator(detail::channel_base<void> const *c)
channel_iterator(receive_channel<void> const *c)

Private Types

typedef hpx::util::iterator_facade<channel_iterator<void>, util::unused_type const, std::input_iterator_tag>

Private Functions

bool get_checked()

bool equal(channel_iterator const &rhs) const
void increment() const

base_type::reference dereference() const
Private Members

hpx::intrusive_ptr<detail::channel_impl_base<util::unused_type>> channel_
bool data_

Friends

friend hpx::lcos::local::hpx::util::iterator_core_access
template <typename T>
class one_element_channel : protected hpx::lcos::local::detail::channel_base<T>

Public Types

typedef T value_type

Public Functions

one_element_channel()

Private Types

typedef detail::channel_base<T> base_type

Friends

friend hpx::lcos::local::channel_iterator< T >
friend hpx::lcos::local::receive_channel< T >
friend hpx::lcos::local::send_channel< T >
template <>
template<>
class one_element_channel< void > : protected hpx::lcos::local::detail::channel_base< void >

Public Types

typedef void value_type

Public Functions

one_element_channel()

Private Types

typedef detail::channel_base<void> base_type

Friends

friend hpx::lcos::local::channel_iterator< void >
friend hpx::lcos::local::receive_channel< void >
friend hpx::lcos::local::send_channel< void >
template <typename T>
class receive_channel : protected hpx::lcos::local::detail::channel_base<T>

Public Functions

receive_channel (channel<T> const &c)
receive_channel (one_element_channel<T> const &c)

Private Types

typedef detail::channel_base<T> base_type

Friends

friend hpx::lcos::local::channel_iterator<T> friend hpx::lcos::local::send_channel<T>

template <>
template<>
class receive_channel<void> : protected hpx::lcos::local::detail::channel_base<void>

Public Functions

receive_channel (channel<void> const &c)
receive_channel (one_element_channel<void> const &c)

Private Types

typedef detail::channel_base<void> base_type

Friends

friend hpx::lcos::local::channel_iterator<void> friend hpx::lcos::local::send_channel<void>

template <typename T>
class send_channel : hpx::lcos::local::detail::channel_base<T>

Public Functions

send_channel (channel<T> const &c)
send_channel (one_element_channel<T> const &c)

Private Types

typedef detail::channel_base<T> base_type

template <>
template<>
class send_channel<void> : hpx::lcos::local::detail::channel_base<void>
Public Functions

send_channel (channel<void> const &c)

send_channel (one_element_channel<void> const &c)

Private Types

typedef detail::channel_base<void> base_type

#include <hpx/local_lcos/force_linking.hpp>

namespace hpx

namespace local_lcos

Functions

force_linking_helper &force_linking()

struct force_linking_helper

Public Members

void (*free)(lcos::local::detail::guard_task *task)

#include <hpx/local_lcos/and_gate.hpp>

namespace hpx

namespace lcos

namespace local

struct and_gate: public hpx::lcos::local::base_and_gate<no_mutex>

Public Functions

and_gate (std::size_t count = 0)

and_gate (and_gate &rhs)

and_gate &operator=(and_gate &rhs)

template <typename Lock>
future<void> get_future (Lock &l, std::size_t count = std::size_t(-1), std::size_t *generation_value = nullptr, error_code &ec = hpx::throws)

template <typename Lock>
shared_future<void> get_shared_future (Lock &l, std::size_t count = std::size_t(-1), std::size_t *generation_value = nullptr, error_code &ec = hpx::throws)

template <typename Lock>
bool set (std::size_t which, Lock &l, error_code &ec = throws)

```cpp
template <typename Lock>
void synchronize (std::size_t generation_value, Lock &l, char const *function_name = "and_gate::synchronize", error_code &ec = throws)
```

### Private Types

```cpp
typedef base_and_gate<no_mutex> base_type
```

```cpp
template <typename Mutex = lcos::local::spinlock>
struct base_and_gate
```

### Public Functions

```cpp
base_and_gate (std::size_t count = 0)
```

This constructor initializes the `base_and_gate` object from the the number of participants to synchronize the control flow with.

```cpp
base_and_gate (base_and_gate &&rhs)
```

```cpp
base_and_gate &operator= (base_and_gate &&rhs)
```

```cpp
future<void> get_future (std::size_t count = std::size_t(-1), std::size_t *generation_value = nullptr, error_code &ec = hpx::throws)
```

```cpp
shared_future<void> get_shared_future (std::size_t count = std::size_t(-1), std::size_t *generation_value = nullptr, error_code &ec = hpx::throws)
```

```cpp
bool set (std::size_t which, error_code &ec = throws)
```

```cpp
void synchronize (std::size_t generation_value, char const *function_name = "base_and_gate<>::synchronize", error_code &ec = throws)
```

Wait for the generational counter to reach the requested stage.

```cpp
std::size_t next_generation ()
```

```cpp
std::size_t generation () const
```

### Protected Types

```cpp
typedef Mutex mutex_type
```

### Protected Functions

```cpp
bool trigger_conditions (error_code &ec = throws)
```

```cpp
template <typename OuterLock>
future<void> get_future (OuterLock &outer_lock, std::size_t count = std::size_t(-1),
                          std::size_t *generation_value = nullptr, error_code &ec = hpx::throws)
```

```cpp
get a future allowing to wait for the gate to fire
```

```cpp
template <typename OuterLock>
shared_future<void> get_shared_future (OuterLock &outer_lock, std::size_t count = std::size_t(-1),
                                        std::size_t *generation_value = nullptr, error_code &ec = hpx::throws)
```

```cpp
get a shared future allowing to wait for the gate to fire
```
template <typename OuterLock>
bool set (std::size_t which, OuterLock &outer_lock, error_code &ec = throws)
    Set the data which has to go into the segment which.

bool test_condition (std::size_t generation_value)

template <typename Lock>
void synchronize (std::size_t generation_value, Lock &l, char const *function_name =
    "base_and_gate<>::synchronize", error_code &ec = throws)

template <typename OuterLock, typename Lock>
void init_locked (OuterLock &outer_lock, Lock &l, std::size_t count, error_code &ec =
    throws)

Private Types

typedef std::list<conditional_trigger *> condition_list_type

Private Members

mutex_type mtx_
boost::dynamic_bitset received_segments_
lcos::local::promise<void> promise_
std::size_t generation_
condition_list_type conditions_

struct manage_condition

Public Functions

template<>
manage_condition (base_and_gate &gate, conditional_trigger &cond)

template<>
~manage_condition ()

template <typename Condition>
template<>
future<void> get_future (Condition &&func, error_code &ec = hpx::throws)

Public Members

template<>
base_and_gate &this_

template<>
condition_list_type::iterator it_

#include <hpx/local_lcos/conditional_trigger.hpp>

namespace hpx

    namespace lcos
namespace local

struct conditional_trigger

Public Functions

conditional_trigger()
conditional_trigger(conditional_trigger &&rhs)
conditional_trigger &operator=(conditional_trigger &&rhs)

template <typename Condition>
future<void> get_future(Condition &&func, error_code &ec = hpx::throws)
    get a future allowing to wait for the trigger to fire

void reset()

bool set(error_code &ec = throws)
    Trigger this object.

Private Members

lcos::local::promise<void> promise_
util::function_nonser<bool ()> cond_

topology
#include <compatibility/hpx/util/cache_aligned_data.hpp>
#include <compatibility/hpx/runtime/threads/topology.hpp>
#include <compatibility/hpx/runtime/threads/cpu_mask.hpp>
#include <hpx/topology/topology.hpp>

namespace hpx

namespace threads


Typedefs

using hpx::threads::hwloc_bitmap_ptr = typedef std::shared_ptr<hpx_hwloc_bitmap_wrapper>

Enums

enum hpx_hwloc_membind_policy
    Please see hwloc documentation for the corresponding enums HWLOC_MEMBIND_XXX.
    Values:
    membinder_default = HWLOC_MEMBIND_DEFAULT
    membinder_firsttouch = HWLOC_MEMBIND_FIRSTTOUCH
    membinder_bind = HWLOC_MEMBIND_BIND
membind_interleave = HWLOC_MEMBIND_INTERLEAVE
membind_replicate = HWLOC_MEMBIND_REPLICATE
membind_nexttouch = HWLOC_MEMBIND_NEXTTOUCH
membind_mixed = HWLOC_MEMBIND_MIXED
membind_user = HWLOC_MEMBIND_MIXED + 256

Functions

topology &create_topology()

std::size_t hardware_concurrency()

std::size_t get_memory_page_size()

struct hpx_hwloc_bitmap_wrapper

Public Functions

HPX_NON_COPYABLE (hpx_hwloc_bitmap_wrapper)

hpx_hwloc_bitmap_wrapper()

hpx_hwloc_bitmap_wrapper (void *bmp)

~hpx_hwloc_bitmap_wrapper()

void reset (hwloc_bitmap_t bmp)

operator bool () const

hwloc_bitmap_t get_bmp () const

Private Members

hwloc_bitmap_t bmp_

Friends

std::ostream &operator<< (std::ostream &, hpx_hwloc_bitmap_wrapper const *bmp)

struct topology

Public Functions

topology ()

~topology ()

std::size_t get_socket_number (std::size_t num_thread, error_code& = throws) const

Return the Socket number of the processing unit the given thread is running on.

Parameters

• ec: [in,out] this represents the error status on exit, if this is pre-initialized to hpx::throws the function will throw on error instead.
`std::size_t get_numa_node_number (std::size_t num_thread, error_code& = throws) const`  
Return the NUMA node number of the processing unit the given thread is running on.

**Parameters**

- `ec`: [in,out] this represents the error status on exit, if this is pre-initialized to `hpx::throws` the function will throw on error instead.

`mask_cref_type get_machine_affinity_mask (error_code &ec = throws) const`  
Return a bit mask where each set bit corresponds to a processing unit available to the application.

**Parameters**

- `ec`: [in,out] this represents the error status on exit, if this is pre-initialized to `hpx::throws` the function will throw on error instead.

`mask_type get_service_affinity_mask (mask_cref_type used_processing_units, error_code &ec = throws) const`  
Return a bit mask where each set bit corresponds to a processing unit available to the service threads in the application.

**Parameters**

- `used_processing_units`: [in] This is the mask of processing units which are not available for service threads.
- `ec`: [in,out] this represents the error status on exit, if this is pre-initialized to `hpx::throws` the function will throw on error instead.

`mask_cref_type get_socket_affinity_mask (std::size_t num_thread, error_code &ec = throws) const`  
Return a bit mask where each set bit corresponds to a processing unit available to the given thread inside the socket it is running on.

**Parameters**

- `ec`: [in,out] this represents the error status on exit, if this is pre-initialized to `hpx::throws` the function will throw on error instead.

`mask_cref_type get_numa_node_affinity_mask (std::size_t num_thread, error_code &ec = throws) const`  
Return a bit mask where each set bit corresponds to a processing unit available to the given thread inside the NUMA domain it is running on.

**Parameters**

- `ec`: [in,out] this represents the error status on exit, if this is pre-initialized to `hpx::throws` the function will throw on error instead.

`mask_type get_numa_node_affinity_mask_from_numa_node (std::size_t num_node) const`  
Return a bit mask where each set bit corresponds to a processing unit associated with the given NUMA node.

**Parameters**

- `ec`: [in,out] this represents the error status on exit, if this is pre-initialized to `hpx::throws` the function will throw on error instead.

`mask_cref_type get_core_affinity_mask (std::size_t num_thread, error_code &ec = throws) const`  
Return a bit mask where each set bit corresponds to a processing unit available to the given thread inside the core it is running on.

**Parameters**

- `ec`: [in,out] this represents the error status on exit, if this is pre-initialized to `hpx::throws` the function will throw on error instead.
mask_cref_type get_thread_affinity_mask (std::size_t num_thread, error_code &ec = throws) const
Return a bit mask where each set bit corresponds to a processing unit available to the given thread.

Parameters
• ec: [in,out] this represents the error status on exit, if this is pre-initialized to hpx::throws the function will throw on error instead.

void set_thread_affinity_mask (mask_cref_type mask, error_code &ec = throws) const
Use the given bit mask to set the affinity of the given thread. Each set bit corresponds to a processing unit the thread will be allowed to run on.

Note Use this function on systems where the affinity must be set from inside the thread itself.

Parameters
• ec: [in,out] this represents the error status on exit, if this is pre-initialized to hpx::throws the function will throw on error instead.

mask_type get_thread_affinity_mask_from_lva (void const *lva, error_code &ec = throws) const
Return a bit mask where each set bit corresponds to a processing unit co-located with the memory the given address is currently allocated on.

Parameters
• ec: [in,out] this represents the error status on exit, if this is pre-initialized to hpx::throws the function will throw on error instead.

void print_affinity_mask (std::ostream &os, std::size_t num_thread, mask_cref_type m, const std::string &pool_name) const
Prints the.

Parameters
• m: to os in a human readable form

bool reduce_thread_priority (error_code &ec = throws) const
Reduce thread priority of the current thread.

Parameters
• ec: [in,out] this represents the error status on exit, if this is pre-initialized to hpx::throws the function will throw on error instead.

std::size_t get_number_of.Sockets () const
Return the number of available NUMA domains.

std::size_t get_number_of_numa_nodes () const
Return the number of available NUMA domains.

std::size_t get_number_of_cores () const
Return the number of available cores.

std::size_t get_number_of_pus () const
Return the number of available hardware processing units.

std::size_t get_number_of_numa_node_cores (std::size_t numa) const
Return number of cores in given numa domain.

std::size_t get_number_of_numa_node_pus (std::size_t numa) const
Return number of processing units in a given numa domain.

std::size_t get_number_of_socket_pus (std::size_t socket) const
Return number of processing units in a given socket.
std::size_t get_number_of_core_pus (std::size_t core) const
Return number of processing units in given core.

std::size_t get_number_of_socket_cores (std::size_t socket) const
Return number of cores units in given socket.

std::size_t get_core_number (std::size_t num_thread, error_code& = throws) const
std::size_t get_pu_number (std::size_t num_core, std::size_t num_pu, error_code &ec = throws) const

mask_type get_cpbind_mask (error_code &ec = throws) const
mask_type get_cpbind_mask (std::thread &handle, error_code &ec = throws) const

hwloc_bitmap_ptr cpuset_to_nodeset (mask_cref_type cpuset) const
convert a cpu mask into a numa node mask in hwloc bitmap form

void write_to_log () const

void *allocate (std::size_t len) const
This is equivalent to malloc(), except that it tries to allocate page-aligned memory from the OS.

void *allocate_membind (std::size_t len, hwloc_bitmap_ptr bitmap, hpx_hwloc_membind_policy policy, int flags) const
allocate memory with binding to a numa node set as specified by the policy and flags (see hwloc docs)

threads::mask_type get_area_membind_nodeset (const void *addr, std::size_t len) const

bool set_area_membind_nodeset (const void *addr, std::size_t len, void *nodeset) const

int get_numa_domain (const void *addr) const

void deallocate (void *addr, std::size_t len) const
Free memory that was previously allocated by allocate.

void print_vector (std::ostream &os, std::vector< std::size_t > const &v) const
void print_mask_vector (std::ostream &os, std::vector< mask_type > const &v) const

void print_hwloc (std::ostream&) const

mask_type init_socket_affinity_mask_from_socket (std::size_t num_socket) const

mask_type init_numa_node_affinity_mask_from_numa_node (std::size_t num_numa_node) const

mask_type init_core_affinity_mask_from_core (std::size_t num_core, mask_cref_type default_mask = empty_mask) const

mask_type init_thread_affinity_mask (std::size_t num_thread) const

mask_type init_thread_affinity_mask (std::size_t num_core, std::size_t num_pu) const

hwloc_bitmap_ptr mask_to_bitmap (mask_cref_type mask, hwloc_obj_type_t htype) const

mask_type bitmap_to_mask (hwloc_bitmap_ptr bitmap, hwloc_obj_type_t htype) const
Private Types

using mutex_type = hpx::util::spinlock

Private Functions

std::size_t init_node_number (std::size_t num_thread, hwloc_obj_type_t type)

std::size_t init_socket_number (std::size_t num_thread)

std::size_t init_numa_node_number (std::size_t num_thread)

std::size_t init_core_number (std::size_t num_thread)

void extract_node_mask (hwloc_obj_t parent, mask_type &mask) const

std::size_t extract_node_count (hwloc_obj_t parent, hwloc_obj_type_t type, std::size_t count) const

mask_type init_machine_affinity_mask () const

mask_type init_socket_affinity_mask (std::size_t num_thread) const

mask_type init_numa_node_affinity_mask (std::size_t num_thread) const

mask_type init_core_affinity_mask (std::size_t num_thread) const

void init_num_of_pus ()

Private Members

hwloc_topology_t topo

std::size_t num_of_pus_

mutex_type topo_mtx

std::vector<std::size_t> socket_numbers_

std::vector<std::size_t> numa_node_numbers_

std::vector<std::size_t> core_numbers_

mask_type machine_affinity_mask_

std::vector<mask_type> socket_affinity_masks_

std::vector<mask_type> numa_node_affinity_masks_

std::vector<mask_type> core_affinity_masks_

std::vector<mask_type> thread_affinity_masks_

Private Static Attributes

mask_type empty_mask

std::size_t memory_page_size_

const std::size_t pu_offset = 0

const std::size_t core_offset = 0
Friends

std::size_t get_memory_page_size()

#include <hpx/topology/cpu_mask.hpp>

batch_environments

#include <compatibility/hpx/util/batch_environment.hpp>
#include <compatibility/hpx/util/batch_environments/alps_environment.hpp>
#include <compatibility/hpx/util/batch_environments/pbs_environment.hpp>
#include <compatibility/hpx/util/batch_environments/slurm_environment.hpp>
#include <hpx/batch_environments/alps_environment.hpp>

namespace hpx

    namespace util

        namespace batch_environments

            struct alps_environment

                Public Functions

                alps_environment (std::vector<std::string> &nodelist, bool debug)
                bool valid() const
                std::size_t node_num() const
                std::size_t num_threads() const
                std::size_t num_localities() const

                Private Members

                std::size_t node_num_
                std::size_t num_threads_
                std::size_t num_localities_
                bool valid_

#include <hpx/batch_environments/batch_environment.hpp>

namespace hpx

    namespace util

        struct batch_environment
Public Types

typedef std::map<boost::asio::ip::tcp::endpoint, std::pair<std::string, std::size_t>> node_map_type

Public Functions

batch_environment (std::vector<std::string> &nodelist, bool have_mpi = false, bool debug = false, bool enable = true)

std::string init_from_nodelist (std::vector<std::string> const &nodes, std::string const &agas_host)

std::size_t retrieve_number_of_threads () const

std::size_t retrieve_number_of_localities () const

std::size_t retrieve_node_number () const

std::string host_name () const

std::string host_name (std::string const &def_hpx_name) const

std::string agas_host_name (std::string const &def_agas) const

std::size_t agas_node () const

bool found_batch_environment () const

std::string get_batch_name () const

Public Members

std::string agas_node_

std::size_t agas_node_num_

std::size_t node_num_

std::size_t num_threads_

node_map_type nodes_

std::size_t num_localities_

std::string batch_name_

bool debug_

#include <hpx/batch_environments/pbs_environment.hpp>

namespace hpx

    namespace util

        namespace batch_environments

            struct pbs_environment
Public Functions

```cpp
pbs_environment (std::vector<std::string> &nodelist, bool have_mpi, bool debug)
```

```cpp
bool valid() const
```

```cpp
std::size_t node_num() const
```

```cpp
std::size_t num_threads() const
```

```cpp
std::size_t num_localities() const
```

Private Functions

```cpp
void read_nodefile (std::vector<std::string> &nodelist, bool have_mpi, bool debug)
```

```cpp
void read_nodelist (std::vector<std::string> &nodelist, bool debug)
```

Private Members

```cpp
std::size_t node_num_
```

```cpp
std::size_t num_localities_
```

```cpp
std::size_t num_threads_
```

```cpp
bool valid_
```

```
#include <hpx/batch_environments/slurm_environment.hpp>
```

```cpp
namespace hpx
```

```cpp
namespace util
```

```cpp
namespace batch_environments
```

```cpp
struct slurm_environment
```

Public Functions

```cpp
slurm_environment (std::vector<std::string> &nodelist, bool debug)
```

```cpp
bool valid() const
```

```cpp
std::size_t node_num() const
```

```cpp
std::size_t num_threads() const
```

```cpp
std::size_t num_localities() const
```
Private Functions

void retrieve_number_of_localities (bool debug)
void retrieve_number_of_tasks (bool debug)
void retrieve_nodelist (std::vector<std::string> &nodes, bool debug)
void retrieve_number_of_threads ()

Private Members

std::size_t node_num_
std::size_t num_threads_
std::size_t num_tasks_
std::size_t num_localities_
bool valid_

```cpp
#include <hpx/compute/host.hpp>
#include <hpx/compute/traits.hpp>
#include <hpx/compute/vector.hpp>
namespace hpx

namespace compute

Functions

template <typename T, typename Allocator>
void swap (vector<T, Allocator> &x, vector<T, Allocator> &y)
Effects: x.swap(y);

template <typename T, typename Allocator = std::allocator<T>>
class vector

Public Types

typedef T value_type
Member types (FIXME: add reference to std.
typedef Allocator allocator_type
typedef alloc_traits::access_target access_target
typedef std::size_t size_type
typedef std::ptrdiff_t difference_type
typedef alloc_traits::reference reference
typedef alloc_traits::const_reference const_reference
typedef alloc_traits::pointer pointer
```
typedef alloc_traits::const_pointer const_pointer
typedef detail::iterator<T, Allocator> iterator
typedef detail::iterator<T const, Allocator> const_iterator
typedef detail::reverse_iterator<T, Allocator> reverse_iterator
typedef detail::const_reverse_iterator<T, Allocator> const_reverse_iterator

Public Functions

vector (Allocator const &alloc = Allocator())
vector (size_type count, T const &value, Allocator const &alloc = Allocator())
vector (size_type count, Allocator const &alloc = Allocator())
template <typename InIter, typename Enable = typename std::enable_if< hpx::traits::is_input_iterator<InIter>::value>::type>
vector (InIter first, InIter last, Allocator const &alloc)
vector (vector const &other)
vector (vector const &other, Allocator const &alloc)
vector (vector &other)
vector (vector &other, Allocator const &alloc)
vector (std::initializer_list<T> init, Allocator const &alloc)
~vector ()

vector &operator= (vector const &other)
vector &operator= (vector &other)

allocator_type get_allocator () const
  Returns the allocator associated with the container.

HPX_HOST_DEVICE reference hpx::compute::vector::operator[](size_type pos)
HPX_HOST_DEVICE const reference hpx::compute::vector::operator[](size_type pos)

pointer data ()
  Returns pointer to the underlying array serving as element storage. The pointer is such that range
  [data(); data() + size()) is always a valid range, even if the container is empty (data() is not
dereferenceable in that case).

const_pointer data () const
  Returns pointer to the underlying array serving as element storage. The pointer is such that range
  [data(); data() + size()) is always a valid range, even if the container is empty (data() is not
dereferenceable in that case).

T *device_data () const
  Returns a raw pointer corresponding to the address of the data allocated on the device.

std::size_t size () const

std::size_t capacity () const
bool empty() const
    Returns: size() == 0.

doctoral resize(size_type size)
    Effects: If size <= size(), equivalent to calling pop_back() size() - size times. If size() < size,
           appends size - size() default-inserted elements to the sequence.
           Requires: T shall be MoveInsertable and DefaultInsertable into *this.
           Remarks: If an exception is thrown other than by the move constructor of a non-CopyInsertable
           T there are no effects.

doctoral resize(size_type size, T const &val)
    Effects: If size <= size(), equivalent to calling pop_back() size() - size times. If size() < size,
           appends size - size() copies of val to the sequence.
           Requires: T shall be CopyInsertable into *this.
           Remarks: If an exception is thrown there are no effects.

iterator begin() 

iterator end() 

const_iterator cbegin() const

const_iterator cend() const

const_iterator begin() const

const_iterator end() const

void swap(vector &other)
    Effects: Exchanges the contents and capacity() of *this with that of x.
           Complexity: Constant time.

void clear()
    Effects: Erases all elements in the range [begin(),end()). Destroys all elements in a. Invalidates
           all references, pointers, and iterators referring to the elements of a and may invalidate the past-
           the-end iterator.
           Post: a.empty() returns true.
           Complexity: Linear.

Private Types

typedef traits::allocator_traits<Allocator> alloc_traits

Private Members

size_type size_

size_type capacity_

allocator_type alloc_

pointer data_
#include <hpx/compute/host/default_executor.hpp>

```cpp
template<>
template<>
struct executor_execution_category<compute::host::default_executor>

Public Types

typedef parallel::execution::parallel_execution_tag type

namespace hpx

namespace compute

namespace host

struct default_executor

Public Functions

default_executor (host::target &target)
template <typename F, typename... Ts>
void post (F &&f, Ts&&... ts)
template <typename F, typename... Ts>
hpx::future<void> async_execute (F &&f, Ts&&... ts)

Public Static Functions

template <typename F, typename... Ts>
static void sync_execute (F &&f, Ts&&... ts)
template <typename F, typename Shape, typename... Ts>
static std::vector<hpx::future<void>> bulk_async_execute (F &&f, Shape const &shape, Ts&&... ts)
template <typename F, typename Shape, typename... Ts>
static void bulk_sync_execute (F &&f, Shape const &shape, Ts&&... ts)

namespace parallel

namespace execution

```
#include <hpx/compute/host/numa_domains.hpp>

namespace hpx

namespace compute

namespace host

## Functions

```cpp
std::vector<target> numa_domains()
```

#include <hpx/compute/host/block_allocator.hpp>

namespace hpx

namespace compute

namespace host

```cpp
template <typename T, typename Executor = hpx::parallel::execution::local_priority_queue_attached_executor>
struct block_allocator
```

```cpp
#include <block_allocator.hpp> The block_allocator allocates blocks of memory evenly divided onto the passed vector of targets. This is done by using first touch memory placement. (maybe better methods will be used in the future...);
```

This allocator can be used to write NUMA aware algorithms:

```cpp
typedef hpx::compute::host::block_allocator<int> allocator_type;  
typedef hpx::compute::vector<int, allocator_type> vector_type;
```

```cpp
auto numa_nodes = hpx::compute::host::numa_domains(); std::size_t N = 2048; vector_type v(N, allocator_type(numa_nodes));
```

### Public Types

```cpp
typedef T value_type
```

```cpp
typedef T *pointer
```

```cpp
typedef const T *const_pointer
```

```cpp
typedef T &reference
```

```cpp
typedef T const &const_reference
```

```cpp
typedef std::size_t size_type
```

```cpp
typedef std::ptrdiff_t difference_type
```

```cpp
typedef Executor executor_type
```

```cpp
typedef std::false_type is_always_equal
```

```cpp
typedef std::true_type propagate_on_container_move_assignment
```

```cpp
typedef std::vector<hpx::host::target> target_type
```
Public Functions

block_allocator()

block_allocator(target_type const &targets)

block_allocator(target_type &&targets)

block_allocator(block_allocator const &alloc)

block_allocator(block_allocator &&alloc)

template <typename U>
block_allocator(block_allocator<U> const &alloc)

template <typename U>
block_allocator(block_allocator<U> &&alloc)

block_allocator &operator= (block_allocator const &rhs)

block_allocator &operator= (block_allocator &&rhs)

pointer address(reference x) const

const_pointer address(const_reference x) const

pointer allocate(size_type n, std::allocator<void>::const_pointer hint = nullptr)

void deallocate(pointer p, size_type n)

size_type max_size() const

template <typename U, typename... Args>
void bulk_construct(U *p, std::size_t count, Args&&... args)

template <typename U, typename... Args>
void construct(U *p, Args&&... args)

template <typename U>
void bulk_destroy(U *p, std::size_t count)

template <typename U>
void destroy(U *p)

target_type const &target() const

Private Members

block_executor<executor_type> executor_

template <typename U>
struct rebind

Public Types

template<>

typedef block_allocator<U> other
#include <hpx/compute/host/target.hpp>

namespace hpx

namespace compute

namespace host

struct target

Public Functions

target()
target(hpx::threads::mask_type mask)
target(hpx::id_type const &locality)
target(hpx::id_type const &locality, hpx::threads::mask_type mask)
native_handle_type &native_handle()
native_handle_type const &native_handle() const
hpx::id_type const &get_locality() const
std::pair<std::size_t, std::size_t> num_pus() const
void synchronize() const
hpx::future<>() get_future() const

Public Static Functions

static std::vector<target> get_local_targets()
static hpx::future<>() get_targets(hpx::id_type const &locality)

Private Functions

void serialize(serialization::input_archive &ar, const unsigned int)
void serialize(serialization::output_archive &ar, const unsigned int)

Private Members

native_handle_type handle_
hpx::id_type locality_

Friends

friend hpx::compute::host::hpx::serialization::access
bool operator==(target const &lhs, target const &rhs)

struct native_handle_type
Public Functions

native_handle_type()
native_handle_type(hpx::threads::mask_type mask)

hpx::threads::mask_type &get_device()

hpx::threads::mask_type const &get_device() const

Private Members

hpx::threads::mask_type mask_

Friends

friend hpx::compute::host::target

#include <hpx/compute/host/target_distribution_policy.hpp>

namespace hpx

namespace compute

namespace host

Variables

target_distribution_policy const target_layout
A predefined instance of the target_distribution_policy for localities. It will represent all NUMA domains of the given locality and will place all items to create here.

struct target_distribution_policy
A target_distribution_policy used for CPU bound localities.

Public Types

typedef compute::detail::target_distribution_policy:host::target> base_type

Public Functions

target_distribution_policy()
Default-construct a new instance of a target_distribution_policy. This policy will represent all devices on the current locality.

target_distribution_policy operator() (std::vector<const target_type> const &targets,
std::size_t num_partitions = std::size_t(-1))

Create a new target_distribution_policy representing the given set of targets

Parameters

• targets: [in] The targets the new instances should represent

#include <hpx/compute/host/target_distribution_policy.hpp>

namespace hpx

namespace compute

namespace host

Variables

target_distribution_policy const target_layout
A predefined instance of the target_distribution_policy for localities. It will represent all NUMA domains of the given locality and will place all items to create here.

struct target_distribution_policy
A target_distribution_policy used for CPU bound localities.

Public Types

typedef compute::detail::target_distribution_policy:host::target> base_type

Public Functions

target_distribution_policy()
Default-construct a new instance of a target_distribution_policy. This policy will represent all devices on the current locality.

target_distribution_policy operator() (std::vector<const target_type> const &targets,
std::size_t num_partitions = std::size_t(-1))

Create a new target_distribution_policy representing the given set of targets

Parameters

• targets: [in] The targets the new instances should represent
target_distribution_policy operator() (std::vector<target_type> &&targets, std::size_t num_partitions = std::size_t(-1)) const
Create a new target_distribution_policy representing the given set of targets

Parameters
• targets: [in] The targets the new instances should represent

target_distribution_policy operator() (target_type const &target, std::size_t num_partitions = 1) const
Create a new target_distribution_policy representing the given target

Parameters
• target: [in] The target the new instances should represent

target_distribution_policy operator() (target_type &&&target, std::size_t num_partitions = 1) const
Create a new target_distribution_policy representing the given target

Parameters
• target: [in] The target the new instances should represent

template <typename Component, typename... Ts>
hpx::future<hpx::id_type> create (Ts&&... ts) const
Create one object on one of the localities associated by this policy instance

Note This function is part of the placement policy implemented by this class
Return A future holding the global address which represents the newly created object

Parameters
• ts: [in] The arguments which will be forwarded to the constructor of the new object.

template <typename Component, typename... Ts>
hpx::future<std::vector<bulk_locality_result>> bulk_create (std::size_t count, Ts&&... ts) const
Create multiple objects on the localities associated by this policy instance

Note This function is part of the placement policy implemented by this class
Return A future holding the list of global addresses which represent the newly created objects

Parameters
• count: [in] The number of objects to create
• ts: [in] The arguments which will be forwarded to the constructors of the new objects.

#include <hpx/compute/host/get_targets.hpp>

namespace hpx

namespace compute

namespace host

Functions

std::vector<target> get_local_targets ()

hpx::future<std::vector<target>> get_targets (hpx::id_type const &locality)
#include <hpx/compute/host/block_executor.hpp>

```cpp
template <typename Executor>
template<>
struct executor_execution_category<compute::host::block_executor<Executor>>
```

Public Types

```cpp
typedef parallel::execution::parallel_execution_tag type
```

namespace hpx

namespace compute

namespace host

```cpp
template <typename Executor = hpx::threads::executors::local_priority_queue_attached_executor>
struct block_executor

#include <block_executor.hpp> The block executor can be used to build NUMA aware programs. It will distribute work evenly across the passed targets
```

Template Parameters

- `Executor`: The underlying executor to use

Public Types

```cpp
typedef hpx::parallel::execution::static_chunk_size executor_parameters_type
```

Public Functions

```cpp
block_executor (std::vector<host::target> const &targets)
block_executor (std::vector<host::target> &&targets)
block_executor (block_executor const &other)
block_executor (block_executor &&other)
block_executor &operator= (block_executor const &other)
block_executor &operator= (block_executor &&other)

template <typename F, typename... Ts>
void post (F &&f, Ts&&... ts)

template <typename F, typename... Ts>
hpx::future<typename hpx::util::detail::invoke_deferred_result<F, Ts...>::type> async_execute (F &&f, Ts&&... ts)
```

```cpp
template <typename F, typename... Ts>
hpx::util::detail::invoke_deferred_result<F, Ts...>::type sync_execute (F &f, Ts&&... ts)
```

```cpp
template <typename F, typename Shape, typename... Ts>
```
std::vector<hpx::future<typename parallel::execution::detail::bulk_function_result<F, Shape, Ts...>::type>> bulk_async_execute(F&& f, Shape const& shape, Ts&&... ts);

template <typename F, typename Shape, typename... Ts> parallel::execution::detail::bulk_execute_result<F, Shape, Ts...>::type bulk_sync_execute(F&& f, Shape const& shape, Ts&&... ts);

std::vector<host::target> const &targets() const

Private Functions
void init_executors()

Private Members
std::vector<host::target> targets_
std::atomic<std::size_t> current_
std::vector<Executor> executors_

namespace parallel
namespace execution

template <typename Executor>
template<>
struct executor_execution_category<compute::host::block_executor<Executor>>

Public Types
typedef parallel::execution::parallel_execution_tag type

#include <hpx/compute/host/traits/access_target.hpp>
template <>
template<>
struct access_target<host::target>

Public Types
typedef host::target target_type
Public Static Functions

```cpp
template <typename T>
static T const &read (target_type const &tgt, T const *t)

template <typename T>
static void write (target_type const &tgt, T *dst, T const *src)
```

```cpp
template <>
template<>
struct access_target<host::target>
```

Public Types

```cpp
typedef std::vector<host::target> target_type
```

Public Static Functions

```cpp
template <typename T>
static T const &read (target_type const &tgt, T const *t)

template <typename T>
static void write (target_type const &tgt, T *dst, T const *src)
```

```cpp
namespace hpx

namespace compute

namespace traits

```cpp
template <>
template<>
struct access_target<host::target>
```

Public Types

```cpp
typedef host::target target_type
```

Public Static Functions

```cpp
template <typename T>
static T const &read (target_type const &tgt, T const *t)

template <typename T>
static void write (target_type const &tgt, T *dst, T const *src)
```

```cpp
template <>
template<>
struct access_target<host::target>
```

Public Types

```cpp
typedef std::vector<host::target> target_type
```
Public Static Functions

```cpp
template <typename T>
static T const & read (target_type const &tgt, T const *t)

template <typename T>
static void write (target_type const &tgt, T *dst, T const *src)
```

```cpp
#include <hpx/compute/traits/access_target.hpp>
#include <hpx/compute/traits/allocator_traits.hpp>

namespace hpx

namespace compute

namespace traits

```cpp
template <typename Allocator>
struct allocator_traits : public std::allocator_traits<Allocator>
```

Public Types

```cpp
typedef detail::get_reference_type<Allocator>::type reference

typedef detail::get_const_reference_type<Allocator>::type const_reference

typedef detail::get_target_traits<Allocator>::type access_target

typedef access_target::target_type target_type
```

Public Static Functions

```cpp
static HPX_HOST_DEVICE auto hpx::compute::traits::allocator_traits::target(Allocator const & alloc)

template <typename... Ts>
static HPX_HOST_DEVICE void hpx::compute::traits::allocator_traits::bulk_construct(Allocator & alloc, pointer p, size_type count, Ts &&... vs)

static HPX_HOST_DEVICE void hpx::compute::traits::allocator_traits::bulk_destroy(Allocator & alloc, pointer p, size_type count)
```

Private Types

```cpp
typedef std::allocator_traits<Allocator> base_type
```

```cpp
#include <hpx/compute/serialization/vector.hpp>

namespace hpx

namespace serialization

```cpp
template <typename T, typename Allocator>
void serialize (input_archive & ar, compute::vector<T, Allocator> & v, unsigned)
```

2.9. API reference 693
template <typename T, typename Allocator>
void serialize(output_archive &ar, compute::vector<T, Allocator> const &v, unsigned)

plugin
#include <compatibility/hpx/util/plugin.hpp>
#include <compatibility/hpx/util/plugin/plugin_factory.hpp>
#include <compatibility/hpx/util/plugin/plugin_wrapper.hpp>
#include <compatibility/hpx/util/plugin/dll.hpp>
#include <compatibility/hpx/util/plugin/concrete_factory.hpp>
#include <compatibility/hpx/util/plugin/abstract_factory.hpp>
#include <compatibility/hpx/util/plugin/export_plugin.hpp>
#include <compatibility/hpx/util/plugin/config.hpp>
#include <compatibility/hpx/util/plugin/virtual_constructor.hpp>
#include <hpx/plugin.hpp>
#include <hpx/plugin/plugin_factory.hpp>
namespace hpx
namespace util
namespace plugin

    template <class BasePlugin>
    struct plugin_factory : public hpx::util::plugin::detail::plugin_factory_item<BasePlugin, detail::plugin_factory_item_base, virtual_constructor<BasePlugin>::type>

Public Functions

    plugin_factory (dll &d, std::string const &basename)

Private Types

template<>
using base_type = detail::plugin_factory_item<BasePlugin, detail::plugin_factory_item_base, typename virtual_constructor<BasePlugin>::type>

    template <class BasePlugin>
    struct static_plugin_factory : public hpx::util::plugin::detail::static_plugin_factory_item<BasePlugin, detail::static_plugin_factory_item_base, virtual_constructor<BasePlugin>::type>

Public Functions

    static_plugin_factory (get_plugins_list_type const &f)
Private Types

template<>
using base_type = detail::static_plugin_factory_item<BasePlugin, detail::static_plugin_factory_item_base, typename virtual_constructor<BasePlugin>::type>

#include <hpx/plugin/plugin_wrapper.hpp>

namespace hpx

namespace util

namespace plugin

template<
  typename Wrapped, typename...
  Parameters
>
struct plugin_wrapper:
  public hpx::util::plugin::detail::dll_handle_holder,
  public Wrapped

Public Functions

plugin_wrapper(dll_handle dll, Parameters... parameters)

#include <hpx/plugin/dll.hpp>

Defines

HPX_HAS_DLOPEN

#include <hpx/plugin/concrete_factory.hpp>
#include <hpx/plugin/abstract_factory.hpp>
#include <hpx/plugin/export_plugin.hpp>

Defines

HPX_PLUGIN_NAME_2 (name1, name2)
HPX_PLUGIN_NAME_3 (name, base, cname)
HPX_PLUGIN_LIST_NAME_ (prefix, name, base)
HPX_PLUGIN_EXPORTER_NAME_ (prefix, name, base, cname)
HPX_PLUGIN_EXPORTER_INSTANCE_NAME_ (prefix, name, base, cname)
HPX_PLUGIN_FORCE_LOAD_NAME_ (prefix, name, base)
HPX_PLUGIN_LIST_NAME (name, base)
HPX_PLUGIN_EXPORTER_NAME (name, base, cname)
HPX_PLUGIN_EXPORTER_INSTANCE_NAME (name, base, cname)
HPX_PLUGIN_FORCE_LOAD_NAME (name, base)
HPX_PLUGIN_LIST_NAME_DYNAMIC (name, base)
HPX_PLUGIN_EXPORTER_NAME_DYNAMIC (name, base, cname)
# HPX Plugin Interface

The HPX plugin interface provides a way to load and use plugins dynamically. It supports loading plugins at runtime and running them as if they were part of the main application.

**Definitions**

- `HPX_PLUGIN_EXPORT_API`
- `HPX_PLUGIN_API`
- `HPX_PLUGIN_ARGUMENT_LIMIT`
- `HPX_PLUGIN_SYMBOLS_PREFIX_DYNAMIC`
- `HPX_PLUGIN_SYMBOLS_PREFIX`
- `HPX_PLUGIN_SYMBOLS_PREFIX_DYNAMIC_STR`
- `HPX_PLUGIN_SYMBOLS_PREFIX_STR`

## HPX Plugin Interface

The HPX plugin interface provides a way to load and use plugins dynamically. It supports loading plugins at runtime and running them as if they were part of the main application.

### Includes

```cpp
#include <hpx/plugin/config.hpp>
#include <hpx/plugin/virtual_constructor.hpp>
```

### Typedefs

```cpp
using hpx::util::plugin::shared_ptr = typedef boost::shared_ptr<T>

typedef exported_plugins_type*(HPX_PLUGIN_API* hpx::util::plugin::get_plugins_list_type) ()

typedef exported_plugins_type* HPX_PLUGIN_API hpx::util::plugin::get_plugins_list_np()
```
using hpx::util::plugin::dll_handle = typedef shared_ptr<get_plugins_list_np>

template <typename BasePlugin>
struct virtual_constructor

Public Types

template<>
using type = hpx::util::detail::pack<>

#include <hpx/plugin/traits/plugin_config_data.hpp>
namespace hpx

namespace traits

template <typename Plugin, typename Enable = void>
struct plugin_config_data

Public Static Functions

static char const *call()

algorithms
#include <hpx/traits/segmented_iterator_traits.hpp>
namespace hpx

namespace traits

template <typename Iterator, typename Enable = void>
struct segmented_iterator_traits

Public Types

typedef std::false_type is_segmented_iterator

template <typename Iterator, typename Enable = void>
struct segmented_local_iterator_traits

Public Types

typedef std::false_type is_segmented_local_iterator

typedef Iterator iterator

typedef Iterator local_iterator

typedef Iterator local_raw_iterator
Public Static Functions

```cpp
static local_raw_iterator const &local (local_iterator const &it)
static local_iterator const &remote (local_raw_iterator const &it)
static local_raw_iterator local (local_iterator &it)
static local_iterator remote (local_raw_iterator &&it)
```

```cpp
#include <hpx/traits/is_value_proxy.hpp>
#include <hpx/parallel/datapar.hpp>
#include <hpx/parallel/tagspec.hpp>
#include <hpx/parallel/numeric.hpp>
#include <hpx/parallel/memory.hpp>
#include <hpx/parallel/container_algorithms.hpp>
#include <hpx/parallel/algorithm.hpp>
#include <hpx/parallel/container_algorithms/partition.hpp>
```

```cpp
namespace hpx

namespace parallel

namespace v1

Functions

```cpp
template <typename ExPolicy, typename Rng, typename Pred, typename Proj = util::projection_identity>
util::detail::algorithm_result<ExPolicy, hpx::traits::range_iterator<Rng>::type>::type
partition (ExPolicy &&policy, Rng &&rng, Pred &&pred, Proj &&proj = Proj())
```

Reorders the elements in the range `rng` in such a way that all elements for which the predicate `pred` returns true precede the elements for which the predicate `pred` returns false. Relative order of the elements is not preserved.

The assignments in the parallel `partition` algorithm invoked with an execution policy object of type `sequenced_policy` execute in sequential order in the calling thread.

**Note** Complexity: Performs at most 2 * N swaps, exactly N applications of the predicate and projection, where N = std::distance(begin(rng), end(rng)).

**Template Parameters**
• **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.

• **Rng**: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an forward iterator.

• **Pred**: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of partition requires Pred to meet the requirements of CopyConstructible.

• **Proj**: The type of an optional projection function. This defaults to util::projection_identity

**Parameters**
- **policy**: The execution policy to use for the scheduling of the iterations.
- **rng**: Refers to the sequence of elements the algorithm will be applied to.
- **pred**: Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by the range rng. This is an unary predicate for partitioning the source iterators. The signature of this predicate should be equivalent to:

```cpp
bool pred(const Type &a);
```

The signature does not need to have const&, but the function must not modify the objects passed to it. The type Type must be such that an object of type FwdIter can be dereferenced and then implicitly converted to Type.

• **proj**: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate is invoked.

The assignments in the parallel partition algorithm invoked with an execution policy object of type parallel_policy or parallel_task_policy are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

**Return** The partition algorithm returns a hpx::future<FwdIter> if the execution policy is of type parallel_task_policy and returns FwdIter otherwise. The partition algorithm returns the iterator to the first element of the second group.

**Template Parameters**
- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.

- **Rng**: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an forward iterator.

- **FwdIter2**: The type of the iterator representing the destination range for the elements that satisfy the predicate pred (deduced). This iterator type must meet the requirements of an forward iterator.

- **FwdIter3**: The type of the iterator representing the destination range for the elements that

The assignments in the parallel partition_copy algorithm invoked with an execution policy object of type sequenced_policy execute in sequential order in the calling thread.

**Note** Complexity: Performs not more than N assignments, exactly N applications of the predicate pred, where N = std::distance(begin(rng), end(rng)).

**Template Parameters**
- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.

- **Rng**: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an forward iterator.

- **FwdIter2**: The type of the iterator representing the destination range for the elements that satisfy the predicate pred (deduced). This iterator type must meet the requirements of an forward iterator.

- **FwdIter3**: The type of the iterator representing the destination range for the elements that
don’t satisfy the predicate \textit{pred} (deduced). This iterator type must meet the requirements of an forward iterator.

- **\textit{Pred}**: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of \textit{partition\_copy} requires \textit{Pred} to meet the requirements of \textit{CopyConstructible}.
- **\textit{Proj}**: The type of an optional projection function. This defaults to util::projection_identity

### Parameters

- **\textit{policy}**: The execution policy to use for the scheduling of the iterations.
- **\textit{rng}**: Refers to the sequence of elements the algorithm will be applied to.
- **\textit{dest\_true}**: Refers to the beginning of the destination range for the elements that satisfy the predicate \textit{pred}.
- **\textit{dest\_false}**: Refers to the beginning of the destination range for the elements that don’t satisfy the predicate \textit{pred}.
- **\textit{pred}**: Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by the range \textit{rng}. This is an unary predicate for partitioning the source iterators. The signature of this predicate should be equivalent to:

```cpp
bool pred(const Type &a);
```

The signature does not need to have const&, but the function must not modify the objects passed to it. The type \textit{Type} must be such that an object of type \textit{FwdIter1} can be dereferenced and then implicitly converted to \textit{Type}.

- **\textit{proj}**: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate is invoked.

The assignments in the parallel \textit{partition\_copy} algorithm invoked with an execution policy object of type \textit{parallel\_policy} or \textit{parallel\_task\_policy} are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

### Return

The \textit{partition\_copy} algorithm returns a \texttt{hpx::future<tagged\_tuple<tag::in(InIter), tag::out1(OutIter1), tag::out2(OutIter2)>>} if the execution policy is of type \textit{parallel\_task\_policy} and returns \texttt{tagged\_tuple<tag::in(InIter), tag::out1(OutIter1), tag::out2(OutIter2)>>} otherwise. The \textit{partition\_copy} algorithm returns the tuple of the source iterator \texttt{last}, the destination iterator to the end of the \texttt{dest\_true} range, and the destination iterator to the end of the \texttt{dest\_false} range.

### Include

```cpp
#include <hpx/parallel/container_algorithms/rotate.hpp>
```

### Namespace

```cpp
namespace hpx {
    namespace parallel {
        namespace v1 {
            template <typename ExPolicy, typename Rng>
            util::detail::algorithm_result<ExPolicy, hpx::util::tagged_pair<tag::begin typename hpx::traits::range_iterator<Rng>::type>,
            tag::end typename hpx::traits::range_iterator<Rng>::type>::type
            rotate(ExPolicy &&policy, Rng &&rng, typename hpx::traits::range_iterator<Rng>::type); // type middle
        }
    }
}
```
The assignments in the parallel rotate algorithm invoked with an execution policy object of type sequenced_policy execute in sequential order in the calling thread.

Note Complexity: Linear in the distance between first and last.

Template Parameters
- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- **Rng**: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of a forward iterator.

Parameters
- **policy**: The execution policy to use for the scheduling of the iterations.
- **rng**: Refers to the sequence of elements the algorithm will be applied to.
- **middle**: Refers to the element that should appear at the beginning of the rotated range.

The assignments in the parallel rotate algorithm invoked with an execution policy object of type parallel_policy or parallel_task_policy are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Note The type of dereferenced FwdIter must meet the requirements of MoveAssignable and MoveConstructible.

Return The rotate algorithm returns a hpx::future<tagged_pair<tag::begin(FwdIter), tag::end(FwdIter)> > if the execution policy is of type parallel_task_policy and returns tagged_pair<tag::begin(FwdIter), tag::end(FwdIter)> otherwise. The rotate algorithm returns the iterator equal to pair(first + (last - new_first), last).

```
template <typename ExPolicy, typename Rng, typename OutIter>
util::detail::algorithm_result<ExPolicy, hpx::util::tagged_pair<tag::in<
    typename hpx::traits::range_iterator<Rng>::type>,
    tag::out<
        OutIter>>>::type rotate_copy
```

OutIter>>:type rotate_copy ExPolicy &&policy, Rng &&rng, typename hpx::traits::range_iterator<Rng>::type middle, OutIter dest_firstCopies the elements from the range [first, last), to another range beginning at dest_first in such a way, that the element new_first becomes the first element of the new range and new_first - 1 becomes the last element.

The assignments in the parallel rotate_copy algorithm invoked with an execution policy object of type sequenced_policy execute in sequential order in the calling thread.

Note Complexity: Performs exactly last - first assignments.

Template Parameters
- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- **Rng**: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of a forward iterator.
- **OutIter**: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an output iterator.

Parameters
- **policy**: The execution policy to use for the scheduling of the iterations.
- **rng**: Refers to the sequence of elements the algorithm will be applied to.
- **middle**: Refers to the element that should appear at the beginning of the rotated range.
- **dest_first**: Refers to the begin of the destination range.

The assignments in the parallel rotate_copy algorithm invoked with an execution policy object of type parallel_policy or parallel_task_policy are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The rotate_copy algorithm returns a hpx::future<tagged_pair<tag::in(FwdIter), tag::out(OutIter)>> > if the execution policy is of type parallel_task_policy and returns tagged_pair<tag::in(FwdIter), tag::out(OutIter)> otherwise. The rotate_copy algorithm re-
turns the output iterator to the element past the last element copied.

```cpp
#include <hpx/parallel/container_algorithms/all_any_none.hpp>

namespace hpx

namespace parallel

namespace v1

Functions

template <typename ExPolicy, typename Rng, typename F, typename Proj = util::projection_identity>
util::detail::algorithm_result<ExPolicy, bool>::type none_of(
    ExPolicy && policy, Rng && rng, F && f, Proj && proj = Proj())

Checks if unary predicate f returns true for no elements in the range rng.

The application of function objects in parallel algorithm invoked with an execution policy object of type sequenced_policy execute in sequential order in the calling thread.

Note Complexity: At most std::distance(begin(rng), end(rng)) applications of the predicate f

Template Parameters

- ExPolicy: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it applies user-provided function objects.
- Rng: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- F: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of none_of requires F to meet the requirements of CopyConstructible.
- Proj: The type of an optional projection function. This defaults to util::projection_identity

Parameters

- policy: The execution policy to use for the scheduling of the iterations.
- rng: Refers to the sequence of elements the algorithm will be applied to.
- f: Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by [first, last). The signature of this predicate should be equivalent to:

```cpp
bool pred(const Type &a);
```

The signature does not need to have const&, but the function must not modify the objects passed to it. The type Type must be such that an object of type FwdIter can be dereferenced and then implicitly converted to Type.
- proj: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate is invoked.

The application of function objects in parallel algorithm invoked with an execution policy object of type parallel_policy or parallel_task_policy are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The none_of algorithm returns a hpx::future<bool> if the execution policy is of type sequenced_task_policy or parallel_task_policy and returns bool otherwise. The none_of algorithm returns true if the unary predicate f returns true for no elements in the range, false otherwise. It returns true if the range is empty.
Checks if unary predicate \( f \) returns true for at least one element in the range \( \text{rng} \).

The application of function objects in parallel algorithm invoked with an execution policy object of type sequenced_policy execute in sequential order in the calling thread.  

**Note** Complexity: At most std::distance(begin(rng), end(rng)) applications of the predicate \( f \)

**Template Parameters**

- \( \text{ExPolicy} \): The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it applies user-provided function objects.
- \( \text{Rng} \): The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- \( \text{F} \): The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of none_of requires \( \text{F} \) to meet the requirements of CopyConstructible.
- \( \text{Proj} \): The type of an optional projection function. This defaults to util::projection_identity

**Parameters**

- \( \text{policy} \): The execution policy to use for the scheduling of the iterations.
- \( \text{rng} \): Refers to the sequence of elements the algorithm will be applied to.
- \( \text{f} \): Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by \([\text{first}, \text{last})\). The signature of this predicate should be equivalent to:

```cpp
bool pred(const Type &a);
```

The signature does not need to have const&, but the function must not modify the objects passed to it. The type \( \text{Type} \) must be such that an object of type \( \text{FwdIter} \) can be dereferenced and then implicitly converted to \( \text{Type} \).
- \( \text{proj} \): Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate is invoked.

The application of function objects in parallel algorithm invoked with an execution policy object of type parallel_policy or parallel_task_policy are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

**Return** The any_of algorithm returns a hpx::future<bool> if the execution policy is of type sequenced_task_policy or parallel_task_policy and returns bool otherwise. The any_of algorithm returns true if the unary predicate \( f \) returns true for at least one element in the range, false otherwise. It returns false if the range is empty.

**Template Parameters**

- \( \text{ExPolicy} \): The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it applies user-provided function objects.
- \( \text{Rng} \): The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- \( \text{F} \): The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of none_of requires \( \text{F} \) to meet the requirements of CopyConstructible.
- \( \text{Proj} \): The type of an optional projection function. This defaults to util::projection_identity

**Parameters**

- \( \text{policy} \): The execution policy to use for the scheduling of the iterations.
- \( \text{rng} \): Refers to the sequence of elements the algorithm will be applied to.
- \( \text{f} \): Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by \([\text{first}, \text{last})\). The signature of this predicate should be equivalent to:

```cpp
bool pred(const Type &a);
```

The signature does not need to have const&, but the function must not modify the objects passed to it. The type \( \text{Type} \) must be such that an object of type \( \text{FwdIter} \) can be dereferenced and then implicitly converted to \( \text{Type} \).
- \( \text{proj} \): Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate is invoked.

The application of function objects in parallel algorithm invoked with an execution policy object of type sequenced_policy execute in sequential order in the calling thread.  

**Note** Complexity: At most std::distance(begin(rng), end(rng)) applications of the predicate \( f \)
• **policy**: The execution policy to use for the scheduling of the iterations.
• **rng**: Refers to the sequence of elements the algorithm will be applied to.
• **f**: Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by [first, last). The signature of this predicate should be equivalent to:

```cpp
bool pred(const Type &a);
```

The signature does not need to have const&, but the function must not modify the objects passed to it. The type `Type` must be such that an object of type `FwdIter` can be dereferenced and then implicitly converted to `Type`.
• **proj**: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate is invoked.

The application of function objects in parallel algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

**Return** The `all_of` algorithm returns a `hpx::future<bool>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `bool` otherwise. The `all_of` algorithm returns true if the unary predicate `f` returns true for all elements in the range, false otherwise. It returns true if the range is empty.

```cpp
#include <hpx/parallel/container_algorithms/find.hpp>

namespace hpx
{
namespace parallel
{
namespace v1
{

Functions

```cpp
template <typename ExPolicy, typename Rng, typename Rng2, typename Pred = detail::equal_to, typename Proj = util::projection_identity>
util::detail::algorithm_result<ExPolicy, typename hpx::traits::range_iterator<Rng>::type>::type
find_end(ExPolicy &&policy, Rng &&rng, Rng2 &&rng2, Pred &&op = Pred(), Proj &&proj = Proj())
```

Returns the last subsequence of elements `rng2` found in the range `rng` using the given predicate `f` to compare elements.

The comparison operations in the parallel `find_end` algorithm invoked with an execution policy object of type `sequenced_policy` execute in sequential order in the calling thread.
Note Complexity: at most $S*(N-S+1)$ comparisons where $S = \text{distance}(\text{begin(rng2)}, \text{end(rng2)})$
and $N = \text{distance}(\text{begin(rng)}, \text{end(rng)})$.

Template Parameters

- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- **Rng**: The type of the first source range (deduced). The iterators extracted from this range type must meet the requirements of a forward iterator.
- **Rng2**: The type of the second source range (deduced). The iterators extracted from this range type must meet the requirements of a forward iterator.
- **Pred**: The type of an optional function/function object to use. Unlike its sequential form, the parallel overload of `replace` requires `Pred` to meet the requirements of `CopyConstructible`. This defaults to `std::equal_to<>`
- **Proj**: The type of an optional projection function. This defaults to `util::projection_identity`

Parameters

- **policy**: The execution policy to use for the scheduling of the iterations.
- **rng**: Refers to the first sequence of elements the algorithm will be applied to.
- **rng2**: Refers to the second sequence of elements the algorithm will be applied to.
- **op**: The binary predicate which returns `true` if the elements should be treated as equal. The signature should be equivalent to the following:

```cpp
template<typename ExPolicy, typename Rng1, typename Rng2, typename Pred = detail::equal_to, typename Proj1 = util::projection_identity, typename Proj2 = util::projection_identity>
```

The signature does not need to have `const &`, but the function must not modify the objects passed to it. The types `Type1` and `Type2` must be such that objects of types `iterator_t<Rng>` and `iterator_t<Rng2>` can be dereferenced and then implicitly converted to `Type1` and `Type2` respectively.
- **proj**: Specifies the function (or function object) which will be invoked for each of the elements of type dereferenced `iterator_t<Rng>` and dereferenced `iterator_t<Rng2>` as a projection operation before the function `op` is invoked.

The comparison operations in the parallel `find_end` algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

This overload of `find_end` is available if the user decides to provide the algorithm their own predicate `op`.

Return The `find_end` algorithm returns a `hpx::future<iterator_t<Rng>>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `iterator_t<Rng>` otherwise. The `find_end` algorithm returns an iterator to the beginning of the last subsequence `rng2` in range `rng`. If the length of the subsequence `rng2` is greater than the length of the range `rng`, `end(rng)` is returned. Additionally if the size of the subsequence is empty or no subsequence is found, `end(rng)` is also returned.
util::detail::algorithm_result<ExPolicy, typename hpx::traits::range_iterator<Rng1>::type>::type find_first_of(
    ExPolicy &&policy, Rng1 &&rng1, Rng2 &&rng2, Pred &&op = Pred(), Proj1 &&proj1 = Proj1(), Proj2 &&proj2 = Proj2())

Searches the range \textit{rng1} for any elements in the range \textit{rng2}. Uses binary predicate \textit{p} to compare elements.

The comparison operations in the parallel \textit{find\_first\_of} algorithm invoked with an execution policy object of type \textit{sequenced\_policy} execute in sequential order in the calling thread.

\textbf{Note} Complexity: at most (S*N) comparisons where \(S = \text{distance}(\text{begin}(\text{rng2}), \text{end}(\text{rng2}))\) and \(N = \text{distance}(\text{begin}(\text{rng1}), \text{end}(\text{rng1}))\).

\textbf{Template Parameters}

- \textit{ExPolicy}: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- \textit{Rng1}: The type of the first source range (deduced). The iterators extracted from this range type must meet the requirements of a forward iterator.
- \textit{Rng2}: The type of the second source range (deduced). The iterators extracted from this range type must meet the requirements of a forward iterator.
- \textit{Pred}: The type of an optional function/function object to use. Unlike its sequential form, the parallel overload of \textit{replace} requires \textit{Pred} to meet the requirements of \textit{CopyConstructible}. This defaults to std::equal_to<>
- \textit{Proj1}: The type of an optional projection function. This defaults to util::projection_identity and is applied to the elements in \textit{rng1}.
- \textit{Proj2}: The type of an optional projection function. This defaults to util::projection_identity and is applied to the elements in \textit{rng2}.

\textbf{Parameters}

- \textit{policy}: The execution policy to use for the scheduling of the iterations.
- \textit{rng1}: Refers to the first sequence of elements the algorithm will be applied to.
- \textit{rng2}: Refers to the second sequence of elements the algorithm will be applied to.
- \textit{op}: The binary predicate which returns \textit{true} if the elements should be treated as equal. The signature should be equivalent to the following:

\begin{verbatim}
bool pred(const Type1 &a, const Type2 &b);
\end{verbatim}

The signature does not need to have \textit{const} \&, but the function must not modify the objects passed to it. The types \textit{Type1} and \textit{Type2} must be such that objects of types iterator$_i<$Rng1$>$ and iterator$_i<$Rng2$>$ can be dereferenced and then implicitly converted to \textit{Type1} and \textit{Type2} respectively.

- \textit{proj1}: Specifies the function (or function object) which will be invoked for each of the
elements of type dereferenced `iterator_t<Rng>` before the function `op` is invoked.

- **proj2**: Specifies the function (or function object) which will be invoked for each of the elements of type dereferenced `iterator_t<Rng2>` before the function `op` is invoked.

The comparison operations in the parallel `find_first_of` algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

This overload of `find_first_of` is available if the user decides to provide the algorithm their own predicate `op`.

**Return** The `find_end` algorithm returns a `hpx::future<iterator_t<Rng>>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `iterator_t<Rng>` otherwise. The `find_first_of` algorithm returns an iterator to the first element in the range `rng1` that is equal to an element from the range `rng2`. If the length of the subsequence `rng2` is greater than the length of the range `rng1`, `end(rng1)` is returned. Additionally if the size of the subsequence is empty or no subsequence is found, `end(rng1)` is also returned.

```cpp
#include <hpx/parallel/container_algorithms/for_each.hpp>

namespace hpx
{

namespace parallel
{

namespace v1
{

Functions

```template<
    typename ExPolicy,
    typename Rng,
    typename F,
    typename Proj = util::projection_identity>

util::detail::algorithm_result<ExPolicy, Rng, hpx::traits::range_iterator<Rng>::type>::type
for_each(ExPolicy&& policy, Rng&& rng, F&& f, Proj&& proj = Proj())
```

Applies `f` to the result of dereferencing every iterator in the given range `rng`.

If `f` returns a result, the result is ignored.

**Note** Complexity: Applies `f` exactly `size(rng)` times.

If the type of `first` satisfies the requirements of a mutable iterator, `f` may apply non-constant functions through the dereferenced iterator.

Unlike its sequential form, the parallel overload of `for_each` does not return a copy of its `Function` parameter, since parallelization may not permit efficient state accumulation.

The application of function objects in parallel algorithm invoked with an execution policy object of type `sequenced_policy` execute in sequential order in the calling thread.

**Template Parameters**

- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it applies user-provided function objects.
- Rng: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- F: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of for_each requires F to meet the requirements of CopyConstructible.
- Proj: The type of an optional projection function. This defaults to util::projection_identity

Parameters
- policy: The execution policy to use for the scheduling of the iterations.
- rng: Refers to the sequence of elements the algorithm will be applied to.
- f: Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by [first, last). The signature of this predicate should be equivalent to:

\[
\langle \text{ignored} \rangle \text{ pred(const Type } &a); 
\]

The signature does not need to have const&. The type Type must be such that an object of type InIter can be dereferenced and then implicitly converted to Type.
- proj: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate is invoked.

The application of function objects in parallel algorithm invoked with an execution policy object of type parallel_policy or parallel_task_policy are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The for_each algorithm returns a hpx::future<InIter> if the execution policy is of type sequenced_task_policy or parallel_task_policy and returns InIter otherwise. It returns last.

#include <hpx/parallel/container_algorithms/reverse.hpp>

namespace hpx

  namespace parallel

  namespace v1

Functions

  template <typename ExPolicy, typename Rng>
  util::detail::algorithm_result<ExPolicy, hpx::traits::range_iterator<Rng>::type>::type reverse(ExPolicy &&policy, Rng &&rng)

Reverses the order of the elements in the range [first, last). Behaves as if applying std::iter_swap to every pair of iterators first+i, (last-i) - 1 for each non-negative i < (last-first)/2.

The assignments in the parallel reverse algorithm invoked with an execution policy object of type sequenced_policy execute in sequential order in the calling thread.

Note Complexity: Linear in the distance between first and last.

Template Parameters
- ExPolicy: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- Rng: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of a bidirectional iterator.

Parameters
• policy: The execution policy to use for the scheduling of the iterations.
• rng: Refers to the sequence of elements the algorithm will be applied to.

The assignments in the parallel reverse algorithm invoked with an execution policy object of type parallel_policy or parallel_task_policy are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The reverse algorithm returns a hpx::future<BidirIter> if the execution policy is of type sequenced_task_policy or parallel_task_policy and returns BidirIter otherwise. It returns last.

```cpp
template <typename ExPolicy, typename Rng, typename OutIter>
util::detail::algorithm_result<ExPolicy,
    hpx::util::tagged_pair<tag::in
        (typename hpx::traits::range_iterator<Rng>::type),
        tag::out
    OutIter>>::type
util::reverse_copy(ExPolicy &&policy, Rng &&rng, OutIter dest_first)
```

Copies the elements from the range [first, last) to another range beginning at dest_first in such a way that the elements in the new range are in reverse order. Behaves as if by executing the assignment *(dest_first + (last - first) - 1 - i) = *(first + i) once for each non-negative i < (last - first) If the source and destination ranges (that is, [first, last) and [dest_first, dest_first+(last-first)) respectively) overlap, the behavior is undefined.

The assignments in the parallel reverse_copy algorithm invoked with an execution policy object of type sequenced_policy execute in sequential order in the calling thread.

Note Complexity: Performs exactly last - first assignments.

Template Parameters
• ExPolicy: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
• Rng: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of a bidirectional iterator.
• OutputIter: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an output iterator.

Parameters
• policy: The execution policy to use for the scheduling of the iterations.
• rng: Refers to the sequence of elements the algorithm will be applied to.
• dest_first: Refers to the begin of the destination range.

The assignments in the parallel reverse_copy algorithm invoked with an execution policy object of type parallel_policy or parallel_task_policy are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The reverse_copy algorithm returns a hpx::future<tagged_pair<tag::in(BidirIter), tag::out(OutIter)>> if the execution policy is of type sequenced_task_policy or parallel_task_policy and returns tagged_pair<tag::in(BidirIter), tag::out(OutIter)> otherwise. The copy algorithm returns the pair of the input iterator forwarded to the first element after the last in the input sequence and the output iterator to the element in the destination range, one past the last element copied.

```cpp
#include <hpx/parallel/container_algorithms/search.hpp>

namespace hpx

    namespace parallel

        namespace v1
```

2.9. API reference 709
Functions

```cpp
template <typename ExPolicy, typename Rng1, typename Rng2, typename Pred = detail::equal_to,
          typename Proj1 = util::projection_identity, typename Proj2 = util::projection_identity>
util::detail::algorithm_result<ExPolicy, hpx::traits::range_iterator<Rng1>::type>::type
search(
  ExPolicy&& policy,
  Rng1&& rng1,
  Rng2&& rng2,
  Pred&& op = Pred(),
  Proj1&& proj1 = Proj1(),
  Proj2&& proj2 = Proj2())
```

Searches the range [first, last) for any elements in the range [s_first, s_last). Uses a provided predicate to compare elements.

The comparison operations in the parallel search algorithm invoked with an execution policy object of type sequenced_policy execute in sequential order in the calling thread.

**Note** Complexity: at most (S*N) comparisons where \( S = \text{distance}(s\_\text{first}, s\_\text{last}) \) and \( N = \text{distance}(\text{first}, \text{last}) \).

**Template Parameters**

- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- **Rng1**: The type of the examine range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- **Rng2**: The type of the search range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- **Pred**: The type of an optional function/function object to use. Unlike its sequential form, the parallel overload of adjacent_find requires Pred to meet the requirements of CopyConstructible. This defaults to `std::equal_to<>`.
- **Proj1**: The type of an optional projection function. This defaults to `util::projection_identity` and is applied to the elements of `Rng1`.
- **Proj2**: The type of an optional projection function. This defaults to `util::projection_identity` and is applied to the elements of `Rng2`.

**Parameters**

- **policy**: The execution policy to use for the scheduling of the iterations.
- **rng1**: Refers to the sequence of elements the algorithm will be examining.
- **rng2**: Refers to the sequence of elements the algorithm will be searching for.
- **op**: Refers to the binary predicate which returns true if the elements should be treated as equal. The signature of the function should be equivalent to

```cpp
bool pred(const Type1 &a, const Type2 &b);
```

The signature does not need to have const &, but the function must not modify the objects passed to it. The types `Type1` and `Type2` must be such that objects of types `FwdIter1` and
FwdIter2 can be dereferenced and then implicitly converted to Type1 and Type2 respectively.

- proj1: Specifies the function (or function object) which will be invoked for each of the elements of rng1 as a projection operation before the actual predicate is invoked.
- proj2: Specifies the function (or function object) which will be invoked for each of the elements of rng2 as a projection operation before the actual predicate is invoked.

The comparison operations in the parallel search algorithm invoked with an execution policy object of type parallel_policy or parallel_task_policy are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

**Return** The search algorithm returns a hpx::future<FwdIter> if the execution policy is of type task_execution_policy and returns FwdIter otherwise. The search algorithm returns an iterator to the beginning of the first subsequence [s_first, s_last) in range [first, last). If the length of the subsequence [s_first, s_last) is greater than the length of the range [first, last), last is returned. Additionally if the size of the subsequence is empty first is returned. If no subsequence is found, last is returned.

```cpp
template <typename ExPolicy, typename Rng1, typename Rng2, typename Pred = detail::equal_to,
          typename Proj1 = util::projection_identity,
          typename Proj2 = util::projection_identity>
util::detail::algorithm_result<ExPolicy, hpx::traits::range_iterator<Rng1>::type>::type
search_n (ExPolicy &&policy, Rng1 &&rng1, std::size_t count,
          Rng2 &&rng2, Pred &&op = Pred(), Proj1 &&proj1 = Proj1(), Proj2 &&proj2 = Proj2())
```

Searches the range [first, last) for any elements in the range [s_first, s_last). Uses a provided predicate to compare elements.

The comparison operations in the parallel search algorithm invoked with an execution policy object of type sequenced_policy execute in sequential order in the calling thread.

**Note** Complexity: at most (S*N) comparisons where S = distance(s_first, s_last) and N = distance(first, last).

**Template Parameters**

- ExPolicy: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- Rng1: The type of the examine range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- Rng2: The type of the search range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- Pred: The type of an optional function/function object to use. Unlike its sequential form, the parallel overload of adjacent_find requires Pred to meet the requirements of CopyConstructible. This defaults to std::equal_to<>
• Proj1: The type of an optional projection function. This defaults to `util::projection_identity` and is applied to the elements of `Rng1`.
• Proj2: The type of an optional projection function. This defaults to `util::projection_identity` and is applied to the elements of `Rng2`.

Parameters
• policy: The execution policy to use for the scheduling of the iterations.
• rng1: Refers to the sequence of elements the algorithm will be examining.
• count: The number of elements to apply the algorithm on.
• rng2: Refers to the sequence of elements the algorithm will be searching for.
• op: Refers to the binary predicate which returns true if the elements should be treated as equal. The signature of the function should be equivalent to

```cpp
bool pred(const Type1 &a, const Type2 &b);
```

The signature does not need to have `const &`, but the function must not modify the objects passed to it. The types `Type1` and `Type2` must be such that objects of types `FwdIter1` and `FwdIter2` can be dereferenced and then implicitly converted to `Type1` and `Type2` respectively.
• proj1: Specifies the function (or function object) which will be invoked for each of the elements of `rng1` as a projection operation before the actual predicate is invoked.
• proj2: Specifies the function (or function object) which will be invoked for each of the elements of `rng2` as a projection operation before the actual predicate is invoked.

The comparison operations in the parallel search algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The search algorithm returns a `hpx::future<FwdIter>` if the execution policy is of type `task_execution_policy` and returns `FwdIter` otherwise. The search algorithm returns an iterator to the beginning of the first subsequence `[s_first, s_last)` in range `[first, last)`. If the length of the subsequence `[s_first, s_last)` is greater than the length of the range `[first, last)`, `last` is returned. Additionally if the size of the subsequence is empty `first` is returned. If no subsequence is found, `last` is returned.

#include <hpx/parallel/container_algorithms/fill.hpp>

namespace hpx

namespace parallel

namespace v1

Functions

```cpp
template <typename ExPolicy, typename Rng, typename T>
util::detail::algorithm_result<ExPolicy>::type fill (ExPolicy &&policy, Rng &&rng, T value)
Assigns the given value to the elements in the range `[first, last)`. The comparisons in the parallel fill algorithm invoked with an execution policy object of type `sequenced_policy` execute in sequential order in the calling thread. Note Complexity: Performs exactly `last - first` assignments.

Template Parameters
• ExPolicy: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
• Rng: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
• T: The type of the value to be assigned (deduced).

Parameters
• policy: The execution policy to use for the scheduling of the iterations.
• rng: Refers to the sequence of elements the algorithm will be applied to.
• value: The value to be assigned.

The comparisons in the parallel fill algorithm invoked with an execution policy object of type parallel_policy or parallel_task_policy are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The fill algorithm returns a hpx::future<void> if the execution policy is of type sequenced_task_policy or parallel_task_policy and returns difference_type otherwise (where difference_type is defined by void).

```
#include <hpx/parallel/container_algorithms/merge.hpp>

namespace hpx
```

Assigns the given value value to the first count elements in the range beginning at first if count > 0. Does nothing otherwise.

The comparisons in the parallel fill_n algorithm invoked with an execution policy object of type sequenced_policy execute in sequential order in the calling thread.

Note Complexity: Performs exactly count assignments, for count > 0.

Template Parameters
• ExPolicy: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
• Rng: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
• Size: The type of the argument specifying the number of elements to apply f to.
• T: The type of the value to be assigned (deduced).

Parameters
• policy: The execution policy to use for the scheduling of the iterations.
• rng: Refers to the sequence of elements the algorithm will be applied to.
• count: Refers to the number of elements starting at first the algorithm will be applied to.
• value: The value to be assigned.

The comparisons in the parallel fill_n algorithm invoked with an execution policy object of type parallel_policy or parallel_task_policy are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The fill_n algorithm returns a hpx::future<void> if the execution policy is of type sequenced_task_policy or parallel_task_policy and returns difference_type otherwise (where difference_type is defined by void).

```
#include <hpx/parallel/container_algorithms/merge.hpp>

namespace hpx
```
namespace parallel

namespace v1

Functions

template <typename ExPolicy, typename Rng1, typename Rng2, typename RandIter3, typename Comp = detail::less,
           typename Proj1 = util::projection_identity,
           typename Proj2 = util::projection_identity>
util::detail::algorithm_result<ExPolicy,
                              hpx::util::tagged_tuple<tag::in1,
                                               typename hpx::traits::range_iterator<Rng1>::type,
                                               tag::in2>
  type merge(ExPolicy &&policy, Rng1 &&rng1, Rng2 &&rng2, RandIter3 dest, Comp &&comp = Comp(), Proj1 &&proj1 = Proj1(), Proj2 &&proj2 = Proj2())
Merges two sorted ranges [first1, last1) and [first2, last2) into one sorted range beginning at dest. The order of equivalent elements in the each of original two ranges is preserved. For equivalent elements in the original two ranges, the elements from the first range precede the elements from the second range. The destination range cannot overlap with either of the input ranges.

The assignments in the parallel merge algorithm invoked with an execution policy object of type sequenced_policy execute in sequential order in the calling thread.

Note Complexity: Performs \(O(\text{std::distance(first1, last1)} + \text{std::distance(first2, last2)})\) applications of the comparison \(comp\) and the each projection.

Template Parameters

- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- **Rng1****: The type of the first source range used (deduced). The iterators extracted from this range type must meet the requirements of an random access iterator.
- **Rng2****: The type of the second source range used (deduced). The iterators extracted from this range type must meet the requirements of an random access iterator.
- **RandIter3****: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an random access iterator.
- **Comp**: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of merge requires Comp to meet the requirements of CopyConstructible. This defaults to std::less<>
- **Proj1**: The type of an optional projection function to be used for elements of the first range. This defaults to util::projection_identity
- **Proj2**: The type of an optional projection function to be used for elements of the second range. This defaults to util::projection_identity

Parameters

- **policy**: The execution policy to use for the scheduling of the iterations.
- **rng1**: Refers to the first range of elements the algorithm will be applied to.
- **rng2**: Refers to the second range of elements the algorithm will be applied to.
- **dest**: Refers to the beginning of the destination range.
- **comp**: \(comp\) is a callable object which returns true if the first argument is less than the second, and false otherwise. The signature of this comparison should be equivalent to:

  ```cpp
  bool comp(const Type1 &a, const Type2 &b);
  ```

The signature does not need to have const&, but the function must not modify the objects passed to it. The types Type1 and Type2 must be such that objects of types RandIter1 and RandIter2 can be dereferenced and then implicitly converted to both Type1 and Type2.

- **proj1**: Specifies the function (or function object) which will be invoked for each of the elements of the first range as a projection operation before the actual comparison \(comp\) is
invoked.

- proj2: Specifies the function (or function object) which will be invoked for each of the elements of the second range as a projection operation before the actual comparison \( \text{comp} \) is invoked.

The assignments in the parallel `merge` algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

**Return** The `merge` algorithm returns a `hpx::future<tagged_tuple<tag::in1(RanIter1),
tag::in2(RanIter2), tag::out(RanIter3)>>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `tagged_tuple<tag::in1(RanIter1),
tag::in2(RanIter2), tag::out(RanIter3)>>` otherwise. The `merge` algorithm returns the tuple of the source iterator `last1`, the source iterator `last2`, the destination iterator to the end of the `dest` range.

**Template Parameters**

- \text{ExPolicy}: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- \text{Rng}: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an random access iterator.
- \text{RandIter}: The type of the source iterators used (deduced). This iterator type must meet the requirements of an random access iterator.
- \text{Comp}: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of `inplace_merge` requires \text{Comp} to meet the requirements of `CopyConstructible`. This defaults to `std::less<>
- \text{Proj}: The type of an optional projection function. This defaults to `util::projection_identity`.

**Parameters**

- \text{policy}: The execution policy to use for the scheduling of the iterations.
- \text{rng}: Refers to the range of elements the algorithm will be applied to.
- \text{middle}: Refers to the end of the first sorted range and the beginning of the second sorted range the algorithm will be applied to.
- \text{comp}: `comp` is a callable object which returns true if the first argument is less than the second, and false otherwise. The signature of this comparison should be equivalent to:

```cpp
bool comp(const Type1 &a, const Type2 &b);
```

The signature does not need to have const&, but the function must not modify the objects passed to it. The types `Type1` and `Type2` must be such that objects of types `RandIter` can be dereferenced and then implicitly converted to both `Type1` and `Type2`.
• **proj**: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate is invoked.

The assignments in the parallel `inplace_merge` algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

**Return** The `inplace_merge` algorithm returns a `hpx::future<RandIter>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `RandIter` otherwise. The `inplace_merge` algorithm returns the source iterator `last`.

```cpp
#include <hpx/parallel/container_algorithms/remove.hpp>

namespace hpx
{
    namespace parallel
    {
        namespace v1
        {
            template<typename ExPolicy, typename Rng, typename T, typename Proj = util::projection_identity>
            ut::detail::algorithm_result<ExPolicy, hpx::traits::range_iterator<Rng>::type>::type
                remove(ExPolicy &&policy, Rng &&rng, T const &value, Proj &&proj = Proj())
        }
    }
}
```

Removes all elements satisfying specific criteria from the range `[first, last)` and returns a past-the-end iterator for the new end of the range. This version removes all elements that are equal to `value`.

The assignments in the parallel `remove` algorithm invoked with an execution policy object of type `sequenced_policy` execute in sequential order in the calling thread.

**Note** Complexity: Performs not more than `last - first` assignments, exactly `last - first` applications of the operator `==()` and the projection `proj`.

**Template Parameters**
- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- **Rng**: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an forward iterator.
- **T**: The type of the value to remove (deduced). This value type must meet the requirements of `CopyConstructible`.
- **Proj**: The type of an optional projection function. This defaults to `util::projection_identity`.

**Parameters**
- **policy**: The execution policy to use for the scheduling of the iterations.
- **rng**: Refers to the sequence of elements the algorithm will be applied to.
- **value**: Specifies the value of elements to remove.
proj: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate is invoked.

The assignments in the parallel remove algorithm invoked with an execution policy object of type parallel_policy or parallel_task_policy are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The remove algorithm returns a hpx::future<FwdIter> if the execution policy is of type sequenced_task_policy or parallel_task_policy and returns FwdIter otherwise. The remove algorithm returns the iterator to the new end of the range.

Removes all elements satisfying specific criteria from the range [first, last) and returns a past-the-end iterator for the new end of the range. This version removes all elements for which predicate pred returns true.

The assignments in the parallel remove_if algorithm invoked with an execution policy object of type sequenced_policy execute in sequential order in the calling thread.

Note Complexity: Performs not more than last - first assignments, exactly last - first applications of the predicate pred and the projection proj.

Template Parameters
- ExPolicy: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- Rng: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an forward iterator.
- Pred: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of remove_if requires Pred to meet the requirements of CopyConstructible.
- Proj: The type of an optional projection function. This defaults to util::projection_identity

Parameters
- policy: The execution policy to use for the scheduling of the iterations.
- rng: Refers to the sequence of elements the algorithm will be applied to.
- pred: Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by [first, last). This is an unary predicate which returns true for the required elements. The signature of this predicate should be equivalent to:

```cpp
bool pred(const Type &a);
```

The signature does not need to have const&, but the function must not modify the objects passed to it. The type Type must be such that an object of type FwdIter can be dereferenced and then implicitly converted to Type.

proj: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate is invoked.

The assignments in the parallel remove_if algorithm invoked with an execution policy object of type parallel_policy or parallel_task_policy are permitted to execute in an unordered fashion in
unspecified threads, and indeterminately sequenced within each thread.

**Return** The `remove_if` algorithm returns a `hpx::future<FwdIter>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `FwdIter` otherwise. The `remove_if` algorithm returns the iterator to the new end of the range.

```
#include <hpx/parallel/container_algorithms/minmax.hpp>

namespace hpx

namespace parallel

namespace v1

Functions

template <typename ExPolicy, typename Rng, typename Proj = util::projection_identity, typename F = detail::less>
util::detail::algorithm_result<ExPolicy, typename hpx::traits::range_traits<Rng>::iterator_type>::type
min_element(ExPolicy &&policy, Rng &&rng, F &&f = F(), Proj &&proj = Proj())
```

Finds the smallest element in the range [first, last) using the given comparison function `f`.

The comparisons in the parallel `min_element` algorithm invoked with an execution policy object of type `sequenced_policy` execute in sequential order in the calling thread.

**Note** Complexity: Exactly `max(N-1, 0)` comparisons, where `N = std::distance(first, last)`.

**Template Parameters**
- `ExPolicy`: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- `Rng`: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an forward iterator.
- `F`: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of `min_element` requires `F` to meet the requirements of `CopyConstructible`.
- `Proj`: The type of an optional projection function. This defaults to `util::projection_identity`.

**Parameters**
- `policy`: The execution policy to use for the scheduling of the iterations.
- `rng`: Refers to the sequence of elements the algorithm will be applied to.
- `f`: The binary predicate which returns true if the the left argument is less than the right element. The signature of the predicate function should be equivalent to the following:

```cpp
bool pred(const Type1 &a, const Type1 &b);
```

The signature does not need to have `const &`, but the function must not modify the objects passed to it. The type `Type1` must be such that objects of type `FwdIter` can be dereferenced.
and then implicitly converted to Type1.

- **proj**: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate is invoked.

The comparisons in the parallel min_element algorithm invoked with an execution policy object of type parallel_policy or parallel_task_policy are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

**Return** The min_element algorithm returns a hpx::future<FwdIter> if the execution policy is of type sequenced_task_policy or parallel_task_policy and returns FwdIter otherwise. The min_element algorithm returns the iterator to the smallest element in the range [first, last). If several elements in the range are equivalent to the smallest element, returns the iterator to the first such element. Returns last if the range is empty.

```cpp
template <typename ExPolicy, typename Rng, typename Proj = util::projection_identity, typename F = detail::less>
util::detail::algorithm_result<ExPolicy, typename hpx::traits::range_traits<Rng>::iterator_type>::type
min_element(
    ExPolicy &&policy,
    Rng &&rng,
    F &&f = F(),
    Proj &&proj = Proj())
```

Finds the greatest element in the range [first, last) using the given comparison function \(f\).

The comparisons in the parallel max_element algorithm invoked with an execution policy object of type sequenced_policy execute in sequential order in the calling thread.

**Note** Complexity: Exactly \(\max(N-1, 0)\) comparisons, where \(N = \text{std::distance(first, last)}\).
of type parallel_policy or parallel_task_policy are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

**Return** The max_element algorithm returns a hpx::future<FwdIter> if the execution policy is of type sequenced_task_policy or parallel_task_policy and returns FwdIter otherwise. The max_element algorithm returns the iterator to the smallest element in the range [first, last). If several elements in the range are equivalent to the smallest element, returns the iterator to the first such element. Returns last if the range is empty.

```cpp
template<typename ExPolicy, typename Rng, typename Proj = util::projection_identity, typename F = detail::less>
util::detail::algorithm_result<ExPolicy, hpx::util::tagged_pair<tag::min<typename hpx::traits::range_traits<Rng>::iterator_type>, tag::max<typename hpx::traits::range_traits<Rng>::iterator_type>> minmax_element(ExPolicy&& policy, Rng&& rng, F&& f = F(), Proj&& proj = Proj()) Finds the greatest element in the range [first, last) using the given comparison function f.
```

The comparisons in the parallel minmax_element algorithm invoked with an execution policy object of type sequenced_policy execute in sequential order in the calling thread.

**Note** Complexity: At most max(floor(3/2*(N-1)), 0) applications of the predicate, where N = std::distance(first, last).

**Template Parameters**
- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- **Rng**: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an forward iterator.
- **F**: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of minmax_element requires F to meet the requirements of CopyConstructible.
- **Proj**: The type of an optional projection function. This defaults to util::projection_identity

**Parameters**
- **policy**: The execution policy to use for the scheduling of the iterations.
- **rng**: Refers to the sequence of elements the algorithm will be applied to.
- **f**: The binary predicate which returns true if the the left argument is less than the right element. This argument is optional and defaults to std::less. The signature of the predicate function should be equivalent to the following:

```cpp
bool pred(const Type1 &a, const Type1 &b);
```

The signature does not need to have const &a, but the function must not modify the objects passed to it. The type Type1 must be such that objects of type FwdIter can be dereferenced and then implicitly converted to Type1.

- **proj**: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate is invoked.

The comparisons in the parallel minmax_element algorithm invoked with an execution policy object of type parallel_policy or parallel_task_policy are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

**Return** The minmax_element algorithm returns a hpx::future<tagged_pair<tag::min(FwdIter), tag::max(FwdIter)>> if the execution policy is of type sequenced_task_policy or parallel_task_policy and returns tagged_pair<tag::min(FwdIter), tag::max(FwdIter)> otherwise. The minmax_element algorithm returns a pair consisting of an iterator to the smallest element as the first element and an iterator to the greatest element as the second. Returns std::make_pair(first, first) if the range is empty. If several elements are equivalent to the smallest element, the iterator to the first such element is returned. If several elements are equivalent
to the largest element, the iterator to the last such element is returned.

```cpp
#include <hpx/parallel/container_algorithms/transform.hpp>

namespace hpx

namespace parallel

namespace v1

Functions

template <typename ExPolicy, typename Rng, typename OutIter, typename F, typename Proj = util::projection_identity>
util::detail::algorithm_result<ExPolicy, hpx::util::tagged_pair<tag::in, typename hpx::traits::range_iterator<Rng>::type>, tag::out>
OutIter>>::type transform(ExPolicy &&policy, Rng &&rng, OutIter dest, F &&f, Proj &&proj = Proj()) Applies the given function \( f \) to the given range \( \text{rng} \) and stores the result in another range, beginning at \( \text{dest} \).

The invocations of \( f \) in the parallel transform algorithm invoked with an execution policy object of type sequenced_policy execute in sequential order in the calling thread.

**Note** Complexity: Exactly size(rng) applications of \( f \)

**Template Parameters**
- ExPolicy: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the invocations of \( f \).
- Rng: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- OutIter: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an output iterator.
- F: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of transform requires \( F \) to meet the requirements of CopyConstructible.
- Proj: The type of an optional projection function. This defaults to util::projection_identity

**Parameters**
- policy: The execution policy to use for the scheduling of the iterations.
- rng: Refers to the sequence of elements the algorithm will be applied to.
- dest: Refers to the beginning of the destination range.
- f: Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by [first, last). This is an unary predicate. The signature of this predicate should be equivalent to:

```
Ret fun(const Type &a);
```

The signature does not need to have const\. The type Type must be such that an object of type InIter can be dereferenced and then implicitly converted to Type. The type Ret must be such that an object of type OutIter can be dereferenced and assigned a value of type Ret.
- proj: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate \( f \) is invoked.

The invocations of \( f \) in the parallel transform algorithm invoked with an execution policy object of type parallel_policy or parallel_task_policy are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.
Return The `transform` algorithm returns a `hpx::future<tagged_pair<tag::in(InIter),
tag::out(OutIter)>>` if the execution policy is of type `parallel_task_policy` and returns `tagged_pair<tag::in(InIter), tag::out(OutIter)>>` otherwise. The `transform` algorithm returns a tuple holding an iterator referring to the first element after the input sequence and the output iterator to the element in the destination range, one past the last element copied.

```cpp
template <typename ExPolicy, typename Rng, typename InIter2, typename OutIter, typename F,
          typename Proj1 = util::projection_identity,
          typename Proj2 = util::projection_identity>
util::detail::algorithm_result<ExPolicy, hpx::util::tagged_tuple<tag::in1(hpx::traits::range_iterator<Rng>::type), tag::in2 InIter2, tag::outOutIter>>,::type transform(ExPolicy &&policy, Rng &&rng, InIter2 first2, OutIter dest, F &&f, Proj1 &&proj1 = Proj1(), Proj2 &&proj2 = Proj2())Applies the given function f to pairs of elements from two ranges: one defined by rng and the other beginning at first2, and stores the result in another range, beginning at dest.
```

The invocations of f in the parallel `transform` algorithm invoked with an execution policy object of type `sequenced_policy` execute in sequential order in the calling thread.

**Note** Complexity: Exactly size(rng) applications of f

**Template Parameters**

- `ExPolicy`: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the invocations of f.
- `Rng`: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- `InIter2`: The type of the source iterators for the second range used (deduced). This iterator type must meet the requirements of an input iterator.
- `OutIter`: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an output iterator.
- `F`: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of `transform` requires F to meet the requirements of `CopyConstructible`.
- `Proj1`: The type of an optional projection function to be used for elements of the first sequence. This defaults to `util::projection_identity`
- `Proj2`: The type of an optional projection function to be used for elements of the second sequence. This defaults to `util::projection_identity`

**Parameters**

- `policy`: The execution policy to use for the scheduling of the iterations.
- `rng`: Refers to the sequence of elements the algorithm will be applied to.
- `first2`: Refers to the beginning of the second sequence of elements the algorithm will be applied to.
- `dest`: Refers to the beginning of the destination range.
- `f`: Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by `[first, last)`. This is a binary predicate. The signature of this predicate should be equivalent to:

```cpp
Ret fun(const Type1 &a, const Type2 &b);
```

The signature does not need to have const&. The types `Type1` and `Type2` must be such that objects of types `InIter1` and `InIter2` can be dereferenced and then implicitly converted to `Type1` and `Type2` respectively. The type `Ret` must be such that an object of type `OutIter` can be dereferenced and assigned a value of type `Ret`.

- `proj1`: Specifies the function (or function object) which will be invoked for each of the elements of the first sequence as a projection operation before the actual predicate `f` is invoked.
- `proj2`: Specifies the function (or function object) which will be invoked for each of the elements of the second sequence as a projection operation before the actual predicate `f` is invoked.
The invocations of \( f \) in the parallel \texttt{transform} algorithm invoked with an execution policy object of type \texttt{parallel_policy} or \texttt{parallel_task_policy} are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

\textbf{Return} The \texttt{transform} algorithm returns a \texttt{hpx::future\langle\texttt{tagged_tuple\langle tag::in1(InIter1), tag::in2(InIter2), tag::out(OutIter)\rangle} \rangle} > if the execution policy is of type \texttt{parallel_task_policy} and returns \texttt{tagged_tuple\langle tag::in1(InIter1), tag::in2(InIter2), tag::out(OutIter)\rangle} > otherwise. The \texttt{transform} algorithm returns a tuple holding an iterator referring to the first element after the first input sequence, an iterator referring to the first element after the second input sequence, and the output iterator referring to the element in the destination range, one past the last element copied.

\begin{verbatim}
template <typename ExPolicy, typename Rng1, typename Rng2, typename OutIter, typename F, typename Proj1 = util::projection_identity, typename Proj2 = util::projection_identity>
util::detail::algorithm_result<ExPolicy, hpx::util::tagged_tuple<tag::in1 typename hpx::traits::range_iterator<Rng1>::type, tag::in2 typename hpx::traits::range_iterator<Rng2>::type, tag::outOutIter>::type
transformExPolicy &policy, Rng1 &&rng1, Rng2 &&rng2, OutIter dest, F &&f, Proj1 &&proj1 = Proj1(), Proj2 &&proj2 = Proj2())Applies the given function \( f \) to pairs of elements from two ranges: one defined by \([\text{first1}, \text{last1})\) and the other beginning at \text{first2}, and stores the result in another range, beginning at \text{dest}.
\end{verbatim}

The invocations of \( f \) in the parallel \texttt{transform} algorithm invoked with an execution policy object of type \texttt{sequenced_policy} execute in sequential order in the calling thread.

\textbf{Note} Complexity: Exactly \( \min(\text{last2-first2}, \text{last1-first1}) \) applications of \( f \)

\textbf{Template Parameters}

- \texttt{ExPolicy}: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the invocations of \( f \).
- \texttt{Rng1}: The type of the first source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- \texttt{Rng2}: The type of the second source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- \texttt{OutIter}: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an output iterator.
- \texttt{F}: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of \texttt{transform} requires \( F \) to meet the requirements of \texttt{CopyConstructible}.
- \texttt{Proj1}: The type of an optional projection function to be used for elements of the first sequence. This defaults to \texttt{util::projection_identity}
- \texttt{Proj2}: The type of an optional projection function to be used for elements of the second sequence. This defaults to \texttt{util::projection_identity}

\textbf{Parameters}

- \texttt{policy}: The execution policy to use for the scheduling of the iterations.
- \texttt{rng1}: Refers to the first sequence of elements the algorithm will be applied to.
- \texttt{rng2}: Refers to the second sequence of elements the algorithm will be applied to.
- \texttt{dest}: Refers to the beginning of the destination range.
- \texttt{f}: Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by \([\text{first}, \text{last})\). This is a binary predicate. The signature of this predicate should be equivalent to:

\begin{verbatim}
Ret fun(const Type1 &a, const Type2 &b);
\end{verbatim}

The signature does not need to have \texttt{const\&}. The types \texttt{Type1} and \texttt{Type2} must be such that objects of types \texttt{InIter1} and \texttt{InIter2} can be dereferenced and then implicitly converted to \texttt{Type1} and \texttt{Type2} respectively. The type \texttt{Ret} must be such that an object of type \texttt{OutIter} can be dereferenced and assigned a value of type \texttt{Ret}. 
• **proj1**: Specifies the function (or function object) which will be invoked for each of the elements of the first sequence as a projection operation before the actual predicate \( f \) is invoked.

• **proj2**: Specifies the function (or function object) which will be invoked for each of the elements of the second sequence as a projection operation before the actual predicate \( f \) is invoked.

The invocations of \( f \) in the parallel *transform* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

**Note** The algorithm will invoke the binary predicate until it reaches the end of the shorter of the two given input sequences.

**Return** The *transform* algorithm returns a `hpx::future<tagged_tuple<tag::in1(InIter1), tag::in2(InIter2), tag::out(OutIter)>>` if the execution policy is of type *parallel_task_policy* and returns `tagged_tuple<tag::in1(InIter1), tag::in2(InIter2), tag::out(OutIter)>>` otherwise. The *transform* algorithm returns a tuple holding an iterator referring to the first element \( r \) the first input sequence, an iterator referring to the first element after the second input sequence, and the output iterator referring to the element in the destination range, one past the last element copied.

```cpp
#include <hpx/parallel/container_algorithms/unique.hpp>

namespace hpx
{

namespace parallel
{

namespace v1
{

Functions

template <typename ExPolicy, typename Rng, typename Pred = detail::equal_to, typename Proj = util::projection_identity>
util::detail::algorithm_result<ExPolicy, hpx::traits::range_iterator<Rng>::type>::type unique(ExPolicy &&policy, Rng &&rng, Pred &&pred = Pred(), Proj &&proj = Proj())

Eliminates all but the first element from every consecutive group of equivalent elements from the range \( \text{rng} \) and returns a past-the-end iterator for the new logical end of the range.

The assignments in the parallel *unique* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

**Note** Complexity: Performs not more than \( N \) assignments, exactly \( N - 1 \) applications of the predicate \( \text{pred} \) and no more than twice as many applications of the projection \( \text{proj} \), where \( N = \text{std::distance}(	ext{begin(rng)}, \text{end(rng)}) \).

**Template Parameters**

• **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes
the assignments.

- **Rng**: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an forward iterator.
- **Pred**: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of `unique` requires `Pred` to meet the requirements of `CopyConstructible`. This defaults to `std::equal_to<>`
- **Proj**: The type of an optional projection function. This defaults to `util::projection_identity`

**Parameters**

- **policy**: The execution policy to use for the scheduling of the iterations.
- **rng**: Refers to the sequence of elements the algorithm will be applied to.
- **pred**: Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by `[first, last)`. This is an binary predicate which returns `true` for the required elements. The signature of this predicate should be equivalent to:

```cpp
def bool pred(const Type &a, const Type &b);
```

The signature does not need to have `const&`, but the function must not modify the objects passed to it. The type `Type` must be such that an object of type `FwdIter1` can be dereferenced and then implicitly converted to `Type`.
- **proj**: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate is invoked.

The assignments in the parallel `unique` algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

**Return** The `unique` algorithm returns a `hpx::future<FwdIter>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `FwdIter` otherwise. The `unique` algorithm returns the iterator to the new end of the range.

**Template Parameters**

- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- **Rng**: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an forward iterator.
- **FwdIter2**: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an forward iterator.
- **Pred**: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of `unique_copy` requires `Pred` to meet the requirements of `CopyConstructible`. This defaults to `std::equal_to<>`
- **Proj**: The type of an optional projection function. This defaults to `util::projection_identity`

**Parameters**

- **policy**: The execution policy to use for the scheduling of the iterations.
• **rng**: Refers to the sequence of elements the algorithm will be applied to.
• **dest**: Refers to the beginning of the destination range.
• **pred**: Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by the range `rng`. This is an binary predicate which returns `true` for the required elements. The signature of this predicate should be equivalent to:

```cpp
class bool pred(const Type &a, const Type &b);
```

The signature does not need to have `const&`, but the function must not modify the objects passed to it. The type `Type` must be such that an object of type `FwdIter1` can be dereferenced and then implicitly converted to `Type`.
• **proj**: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate is invoked.

The assignments in the parallel `unique_copy` algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

**Return** The `unique_copy` algorithm returns a `hpx::future<tagged_pair<tag::in(FwdIter1), tag::out(FwdIter2)>>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `tagged_pair<tag::in(FwdIter1), tag::out(FwdIter2)>>` otherwise. The `unique_copy` algorithm returns the pair of the source iterator to `last`, and the destination iterator to the end of the `dest` range.

```
#include <hpx/parallel/container_algorithms/replace.hpp>

namespace hpx

namespace parallel

namespace v1

Functions

```cpp
template <typename ExPolicy, typename Rng, typename T1, typename T2, typename Proj = util::projection_identity>
util::detail::algorithm_result<ExPolicy, hpx::traits::range_traits<Rng>::iterator_type>::type
replace(ExPolicy &&policy, Rng &&rng, T1 const &old_value, T2 const &new_value, Proj &&proj = Proj())
```

Replaces all elements satisfying specific criteria with `new_value` in the range `[first, last)`. Effects: Substitutes elements referred by the iterator `it` in the range `[first,last)` with `new_value`, when the following corresponding conditions hold: `INVOKE(proj, *it) == old_value`
**Note** Complexity: Performs exactly last - first assignments.
The assignments in the parallel replace algorithm invoked with an execution policy object of type sequenced_policy execute in sequential order in the calling thread.

**Template Parameters**
- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- **Rng**: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of a forward iterator.
- **T1**: The type of the old value to replace (deduced).
- **T2**: The type of the new values to replace (deduced).
- **Proj**: The type of an optional projection function. This defaults to util::projection_identity

**Parameters**
- **policy**: The execution policy to use for the scheduling of the iterations.
- **rng**: Refers to the sequence of elements the algorithm will be applied to.
- **old_value**: Refers to the old value of the elements to replace.
- **new_value**: Refers to the new value to use as the replacement.
- **proj**: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate is invoked.

The assignments in the parallel replace algorithm invoked with an execution policy object of type parallel_policy or parallel_task_policy are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

**Return** The replace algorithm returns a hpx::future<void> if the execution policy is of type sequenced_task_policy or parallel_task_policy and returns void otherwise.

```
template <typename ExPolicy, typename Rng, typename F, typename T, typename Proj = util::projection_identity>
util::detail::algorithm_result<ExPolicy, typename hpx::traits::range_traits<Rng>::iterator_type>::type
replace_if(ExPolicy&& policy, Rng&& rng, F&& f, T const& new_value, Proj&& proj = Proj())
```

Replaces all elements satisfying specific criteria (for which predicate f returns true) with new_value in the range [first, last).

**Effects**: Substitutes elements referred by the iterator it in the range [first, last) with new_value, when the following corresponding conditions hold: INVOKE(f, INVOKE(proj, *it)) != false

**Note** Complexity: Performs exactly last - first applications of the predicate.
The assignments in the parallel replace_if algorithm invoked with an execution policy object of type sequenced_policy execute in sequential order in the calling thread.

**Template Parameters**
- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- **Rng**: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of a forward iterator.
- **F**: The type of the function/function object to use (deduced). Unlike its sequential form,
the parallel overload of `equal` requires `F` to meet the requirements of `CopyConstructible`.

- **T**: The type of the new values to replace (deduced).
- **Proj**: The type of an optional projection function. This defaults to `util::projection_identity`

**Parameters**

- **policy**: The execution policy to use for the scheduling of the iterations.
- **rng**: Refers to the sequence of elements the algorithm will be applied to.
- **f**: Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by `[first, last)`. This is a unary predicate which returns `true` for the elements which need to replaced. The signature of this predicate should be equivalent to:

  ```
  bool pred(const Type &a);
  ```

The signature does not need to have `const &`, but the function must not modify the objects passed to it. The type `Type` must be such that an object of type `FwdIter` can be dereferenced and then implicitly converted to `Type`.

- **new_value**: Refers to the new value to use as the replacement.
- **proj**: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate is invoked.

The assignments in the parallel `replace_if` algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

**Return**

The `replace_if` algorithm returns a `hpx::future<FwdIter>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `void` otherwise. It returns `last`.

**Template Parameters**

- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- **Rng**: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- **OutIter**: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an output iterator.
- **T1**: The type of the old value to replace (deduced).
- **T2**: The type of the new values to replace (deduced).
- **Proj**: The type of an optional projection function. This defaults to `util::projection_identity`

**Parameters**

- **policy**: The execution policy to use for the scheduling of the iterations.
- **rng**: Refers to the sequence of elements the algorithm will be applied to.
• dest: Refers to the beginning of the destination range.
• old_value: Refers to the old value of the elements to replace.
• new_value: Refers to the new value to use as the replacement.
• proj: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate is invoked.

The assignments in the parallel replace_copy algorithm invoked with an execution policy object of type parallel_policy or parallel_task_policy are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The replace_copy algorithm returns a hpx::future<tagged_pair<tag::in(InIter), tag::out(OutIter)> > if the execution policy is of type sequenced_task_policy or parallel_task_policy and returns tagged_pair<tag::in(InIter), tag::out(OutIter)> otherwise. The copy algorithm returns the pair of the input iterator last and the output iterator to the element in the destination range, one past the last element copied.

```
template <typename ExPolicy, typename Rng, typename OutIter, typename F, typename T, typename Proj = util::projection_identity>
util::detail::algorithm_result<ExPolicy, hpx::util::tagged_pair<tag::in(hpx::traits::range_traits<Rng>::iterator_type), tag::out> OutIter>::type replace_copy_if
```

```
ExPolicy &&policy, Rng &&rng, OutIter dest, F &&f, T const &new_value, Proj &&proj = Proj()Copies the all elements from the range [first, last) to another range beginning at dest replacing all elements satisfying a specific criteria with new_value.
```

Effects: Assigns to every iterator it in the range [result, result + (last - first)) either new_value or *(first + (it - result)) depending on whether the following corresponding condition holds: INVOKE(f, INVOKE(proj, *(first + (i - result)))) != false

The assignments in the parallel replace_copy_if algorithm invoked with an execution policy object of type sequenced_policy execute in sequential order in the calling thread.

Note Complexity: Performs exactly last - first applications of the predicate.

Template Parameters
• ExPolicy: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
• Rng: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
• OutIter: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an output iterator.
• F: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of equal requires F to meet the requirements of CopyConstructible. (deduced).
• T: The type of the new values to replace (deduced).
• Proj: The type of an optional projection function. This defaults to util::projection_identity

Parameters
• policy: The execution policy to use for the scheduling of the iterations.
• rng: Refers to the sequence of elements the algorithm will be applied to.
• dest: Refers to the beginning of the destination range.
• f: Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by [first, last). This is an unary predicate which returns true for the elements which need to replaced. The signature of this predicate should be equivalent to:

```cpp
bool pred(const Type &a);
```

The signature does not need to have const&, but the function must not modify the objects passed to it. The type Type must be such that an object of type FwdIter can be dereferenced
and then implicitly converted to Type.

- **new_value**: Refers to the new value to use as the replacement.
- **proj**: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate is invoked.

The assignments in the parallel replace_copy_if algorithm invoked with an execution policy object of type parallel_policy or parallel_task_policy are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

**Return** The replace_copy_if algorithm returns a hpx::future<tagged_pair<tag::in(InIter), tag::out(OutIter)>> if the execution policy is of type sequenced_task_policy or parallel_task_policy and returns tagged_pair<tag::in(InIter), tag::out(OutIter)> otherwise. The replace_copy_if algorithm returns the input iterator last and the output iterator to the element in the destination range, one past the last element copied.

```cpp
#include <hpx/parallel/container_algorithms/move.hpp>

namespace hpx
{
namespace parallel
{
namespace v1
{

Functions

template <typename ExPolicy, typename Rng, typename OutIter>
util::detail::algorithm_result<ExPolicy, hpx::util::tagged_pair<tag::in<typename hpx::traits::range_traits<Rng>::iterator_type>, tag::out OutIter>>::type move
OutIter&& policy, Rng &&rng, OutIter dest

Moves the elements in the range rng to another range beginning at dest. After this operation the elements in the moved-from range will still contain valid values of the appropriate type, but not necessarily the same values as before the move.

The assignments in the parallel copy algorithm invoked with an execution policy object of type sequenced_policy execute in sequential order in the calling thread.

**Note** Complexity: Performs exactly std::distance(begin(rng), end(rng)) assignments.

**Template Parameters**
- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- **Rng**: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- **OutIter**: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an output iterator.

**Parameters**
- **policy**: The execution policy to use for the scheduling of the iterations.
- **rng**: Refers to the sequence of elements the algorithm will be applied to.
- **dest**: Refers to the beginning of the destination range.

The assignments in the parallel copy algorithm invoked with an execution policy object of type parallel_policy or parallel_task_policy are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

**Return** The move algorithm returns a hpx::future<tagged_pair<tag::in(iterator_t<Rng>), tag::out(FwdIter2)>> if the execution policy is of type sequenced_task_policy or parallel_task_policy.
lel_task_policy and returns tagged_pair<tag::in(iterator_t<Rng>), tag::out(FwdIter2)> otherwise. The move algorithm returns the pair of the input iterator last and the output iterator to the element in the destination range, one past the last element moved.

```cpp
#include <hpx/parallel/container_algorithms/copy.hpp>

namespace hpx

namespace parallel

namespace v1

Functions

```cpp
template <typename ExPolicy, typename Rng, typename OutIter>
util::detail::algorithm_result<ExPolicy, hpx::util::tagged_pair<tag::in typename hpx::traits::range_traits<Rng>::iterator_type, tag::out OutIter>>::type

OutIter>::type copy
ExPolicy &\&policy, Rng &\&rng, OutIter dest

Copies the elements in the range rng to another range beginning at dest.

The assignments in the parallel copy algorithm invoked with an execution policy object of type sequenced_policy execute in sequential order in the calling thread.

Note Complexity: Performs exactly std::distance(begin(rng), end(rng)) assignments.

Template Parameters

- ExPolicy: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- Rng: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- OutIter: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an output iterator.

Parameters

- policy: The execution policy to use for the scheduling of the iterations.
- rng: Refers to the sequence of elements the algorithm will be applied to.
- dest: Refers to the beginning of the destination range.

The assignments in the parallel copy algorithm invoked with an execution policy object of type parallel_policy or parallel_task_policy are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The copy algorithm returns a hpx:future<tagged_pair<tag::in(iterator_t<Rng>), tag::out(FwdIter2)>> if the execution policy is of type sequenced_task_policy or parallel_task_policy and returns tagged_pair<tag::in(iterator_t<Rng>), tag::out(FwdIter2)> otherwise. The copy algorithm returns the pair of the input iterator last and the output iterator to the element in the destination range, one past the last element moved.

```cpp
template <typename ExPolicy, typename Rng, typename OutIter, typename F, typename Proj = util::projection_identity>
util::detail::algorithm_result<ExPolicy, hpx::util::tagged_pair<tag::in typename hpx::traits::range_traits<Rng>::iterator_type>,

OutIter>::type copy_if
ExPolicy &\&policy, Rng &\&rng, OutIter dest, F &\&f, Proj &\&proj = Proj() Copies the elements in the range rng to another range beginning at dest. Copies only the elements for which the predicate f returns true. The order of the elements that are not removed is preserved.

2.9. API reference 731
The assignments in the parallel `copy_if` algorithm invoked with an execution policy object of type `sequenced_policy` execute in sequential order in the calling thread.

**Note** Complexity: Performs not more than `std::distance(begin(rng), end(rng))` assignments, exactly `std::distance(begin(rng), end(rng))` applications of the predicate `f`.

**Template Parameters**
- `ExPolicy`: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- `Rng`: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- `OutIter`: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an output iterator.
- `F`: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of `copy_if` requires `F` to meet the requirements of `CopyConstructible`.
- `Proj`: The type of an optional projection function. This defaults to `util::projection_identity`

**Parameters**
- `policy`: The execution policy to use for the scheduling of the iterations.
- `rng`: Refers to the sequence of elements the algorithm will be applied to.
- `dest`: Refers to the beginning of the destination range.
- `f`: Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by `[first, last)`. This is an unary predicate which returns `true` for the required elements. The signature of this predicate should be equivalent to:

```cpp
bool pred(const Type &a);
```

The signature does not need to have `const&`, but the function must not modify the objects passed to it. The type `Type` must be such that an object of type `InIter` can be dereferenced and then implicitly converted to `Type`.
- `proj`: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate is invoked.

The assignments in the parallel `copy_if` algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

**Return** The `copy_if` algorithm returns a `hpx::future<tagged_pair<tag::in(iterator_t<Rng>),
tag::out(FwdIter2)>>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `tagged_pair<tag::in(iterator_t<Rng>), tag::out(FwdIter2)>>` otherwise. The `copy_if` algorithm returns the pair of the input iterator `last` and the output iterator to the element in the destination range, one past the last element copied.

```cpp
#include <hpx/parallel/container_algorithms/generate.hpp>

namespace hpx
{

namespace parallel
{

namespace v1
{

Functions

template <typename ExPolicy, typename Rng, typename F>
```
Assign each element in range [first, last) a value generated by the given function object \( f \).

The assignments in the parallel `generate` algorithm invoked with an execution policy object of type `sequenced_policy` execute in sequential order in the calling thread.

**Note** Complexity: Exactly `distance(first, last)` invocations of \( f \) and assignments.

**Template Parameters**
- `ExPolicy`: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- `Rng`: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an forward iterator.
- `F`: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of `equal` requires \( F \) to meet the requirements of `CopyConstructible`.

**Parameters**
- `policy`: The execution policy to use for the scheduling of the iterations.
- `rng`: Refers to the sequence of elements the algorithm will be applied to.
- `f`: generator function that will be called. signature of function should be equivalent to the following:

```cpp
Ret fun();
```

The type `Ret` must be such that an object of type `FwdIter` can be dereferenced and assigned a value of type `Ret`.

The assignments in the parallel `generate` algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

**Return** The `replace_if` algorithm returns a `hpx::future<FwdIter>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `FwdIter` otherwise. It returns `last`.

```cpp
#include <hpx/parallel/container_algorithms/is_heap.hpp>

namespace hpx
{

namespace parallel
{

namespace v1
{

Functions

```cpp
template <typename ExPolicy, typename Rng, typename Comp = detail::less, typename Proj = util::projection_identity>
util::detail::algorithm_result<ExPolicy, typename hpx::traits::range_iterator<Rng>::type>::type generate (ExPolicy &&policy, Rng &&rng, F &&f)
```

Returns whether the range is max heap. That is, true if the range is max heap, false otherwise.

The function uses the given comparison function object `comp` (defaults to using operator\(<\)).
comp has to induce a strict weak ordering on the values.

**Note** Complexity: Performs at most N applications of the comparison comp, at most 2 * N applications of the projection proj, where N = last - first.

**Template Parameters**
- ExPolicy: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- Rng: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an random access iterator.
- Comp: The type of the function/function object to use (deduced).
- Proj: The type of an optional projection function. This defaults to util::projection_identity

**Parameters**
- policy: The execution policy to use for the scheduling of the iterations.
- rng: Refers to the sequence of elements the algorithm will be applied to.
- comp: comp is a callable object. The return value of the INVOKE operation applied to an object of type Comp, when contextually converted to bool, yields true if the first argument of the call is less than the second, and false otherwise. It is assumed that comp will not apply any non-constant function through the dereferenced iterator.
- proj: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate is invoked.

The application of function objects in parallel algorithm invoked with an execution policy object of type sequenced_policy execute in sequential order in the calling thread.

The application of function objects in parallel algorithm invoked with an execution policy object of type parallel_policy or parallel_task_policy are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

**Return** The is_heap algorithm returns a hpx::future<bool> if the execution policy is of type sequenced_task_policy or parallel_task_policy and returns bool otherwise. The is_heap algorithm returns whether the range is max heap. That is, true if the range is max heap, false otherwise.

```template <typename ExPolicy, typename Rng, typename Comp = detail::less, typename Proj = util::projection_identity>
util::detail::algorithm_result<ExPolicy, typename hpx::traits::range_iterator<Rng>::type>::type
is_heap_until(ExPolicy&& policy, Rng&& rng, Comp&& comp = Comp(), Proj&& proj = Proj())```

Returns the upper bound of the largest range beginning at first which is a max heap. That is, the last iterator it for which range [first, it) is a max heap. The function uses the given comparison function object comp (defaults to using operator<()).

comp has to induce a strict weak ordering on the values.

**Note** Complexity: Performs at most N applications of the comparison comp, at most 2 * N applications of the projection proj, where N = last - first.

**Template Parameters**
- ExPolicy: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes
the assignments.

- Rng: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an random access iterator.
- Comp: The type of the function/function object to use (deduced).
- Proj: The type of an optional projection function. This defaults to util::projection_identity

**Parameters**

- policy: The execution policy to use for the scheduling of the iterations.
- rng: Refers to the sequence of elements the algorithm will be applied to.
- comp: comp is a callable object. The return value of the INVOKE operation applied to an object of type Comp, when contextually converted to bool, yields true if the first argument of the call is less than the second, and false otherwise. It is assumed that comp will not apply any non-constant function through the dereferenced iterator.
- proj: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate is invoked.

The application of function objects in parallel algorithm invoked with an execution policy object of type sequenced_policy execute in sequential order in the calling thread.

The application of function objects in parallel algorithm invoked with an execution policy object of type parallel_policy or parallel_task_policy are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

**Return**

The is_heap_until algorithm returns a hpx::future<RandIter> if the execution policy is of type sequenced_task_policy or parallel_task_policy and returns RandIter otherwise. The is_heap_until algorithm returns the upper bound of the largest range beginning at first which is a max heap. That is, the last iterator it for which range [first, it) is a max heap.

```cpp
#include <hpx/parallel/container_algorithms/remove_copy.hpp>

namespace hpx
{

namespace parallel
{

namespace v1
{

Functions

```template<typename ExPolicy, typename Rng, typename OutIter, typename T, typename Proj = util::projection_identity>
type remove_copy(ExPolicy &&policy, Rng &&rng, OutIter dest, T const &val, Proj &&proj = Proj())
```

Copies the elements in the range, defined by [first, last), to another range beginning at dest. Copies only the elements for which the comparison operator returns false when compare to val. The order of the elements that are not removed is preserved.

Effects: Copies all the elements referred to by the iterator it in the range [first,last) for which the following corresponding conditions do not hold: INVOKE(proj, *it) == value

The assignments in the parallel remove_copy algorithm invoked with an execution policy object of type sequenced_policy execute in sequential order in the calling thread.

**Note** Complexity: Performs not more than last - first assignments, exactly last - first applications of the predicate f.

**Template Parameters**
• ExPolicy: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
• Rng: The type of the source range used (deduced). The iterators extracted from this range must meet the requirements of an input iterator.
• OutIter: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an output iterator.
• T: The type that the result of dereferencing InIter is compared to.
• Proj: The type of an optional projection function. This defaults to util::projection_identity

Parameters
• policy: The execution policy to use for the scheduling of the iterations.
• rng: Refers to the sequence of elements the algorithm will be applied to.
• dest: Refers to the beginning of the destination range.
• val: Value to be removed.
• proj: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate is invoked.

The assignments in the parallel `remove_copy` algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return
The `remove_copy` algorithm returns a `hpx::future<tagged_pair<tag::in(InIter), tag::out(OutIter)>>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `tagged_pair<tag::in(InIter), tag::out(OutIter)>` otherwise. The `copy` algorithm returns the pair of the input iterator forwarded to the first element after the last in the input sequence and the output iterator to the element in the destination range, one past the last element copied.

```template <typename ExPolicy, typename Rng, typename OutIter, typename F, typename Proj = util::projection_identity>
util::detail::algorithm_result<ExPolicy, hpx::util::tagged_pair<tag::in<typename hpx::traits::range_traits<Rng>::iterator_type>, tag::out OutIter>>::type
remove_copy_if(ExPolicy &&policy, Rng &&rng, OutIter dest, F &&f, Proj &&proj = Proj())
```

Copies the elements in the range, defined by `[first, last)`, for which the following corresponding conditions do not hold: `INVOKE(pred, INVOKE(proj, *it)) != false`. The order of the elements that are not removed is preserved.

Effects: Copies all the elements referred to by the iterator it in the range `[first,last)` for which the following corresponding conditions do not hold: `INVOKE(pred, INVOKE(proj, *it)) != false`. The order of the elements is preserved.

The assignments in the parallel `remove_copy_if` algorithm invoked with an execution policy object of type `sequenced_policy` execute in sequential order in the calling thread.

Note
Complexity: Performs not more than `last - first` assignments, exactly `last - first` applications of the predicate `f`.

Template Parameters
• ExPolicy: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
• Rng: The type of the source range used (deduced). The iterators extracted from this range must meet the requirements of an input iterator.
• OutIter: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an output iterator.
• F: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of `copy_if` requires `F` to meet the requirements of `CopyConstructible`.
• Proj: The type of an optional projection function. This defaults to `util::projection_identity`
• **rng**: Refers to the sequence of elements the algorithm will be applied to.
• **dest**: Refers to the beginning of the destination range.
• **f**: Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by \([\text{first}, \text{last})\). This is an unary predicate which returns `true` for the elements to be removed. The signature of this predicate should be equivalent to:

```cpp
bool pred(const Type &a);
```

The signature does not need to have `const&`, but the function must not modify the objects passed to it. The type `Type` must be such that an object of type `InIter` can be dereferenced and then implicitly converted to `Type`.
• **proj**: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate is invoked.

The assignments in the parallel `remove_copy_if` algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

**Return** The `remove_copy_if` algorithm returns a `hpx::future<tagged_pair<tag::in(InIter), tag::out(OutIter)>>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `tagged_pair<tag::in(InIter), tag::out(OutIter)>>` otherwise. The `copy` algorithm returns the pair of the input iterator forwarded to the first element after the last in the input sequence and the output iterator to the element in the destination range, one past the last element copied.

```cpp
#include <hpx/parallel/container_algorithms/sort.hpp>

namespace hpx

namespace parallel

namespace v1

Functions

```cpp
template <typename ExPolicy, typename Rng, typename Proj = util::projection_identity, typename Compare = detail::less>
util::detail::algorithm_result<ExPolicy, tagged_pair<tag::in(Rng), tag::out(OutIter)>> sort(
    ExPolicy &&policy,
    Rng &&rng,
    Compare &&comp = Compare(),
    Proj &&proj = Proj())
```
A sequence is sorted with respect to a comparator \texttt{comp} and a projection \texttt{proj} if for every iterator \(i\) pointing to the sequence and every non-negative integer \(n\) such that \(i + n\) is a valid iterator pointing to an element of the sequence, and \(\text{INVOKE}(	ext{comp}, \text{INVOKE}(	ext{proj}, *(i + n)), \text{INVOKE}(	ext{proj}, *i)) == \text{false}\).

\textbf{Note} Complexity: \(O(N\log(N))\), where \(N = \text{std::distance}(	ext{begin(rng), end(rng)})\) comparisons. \texttt{comp} has to induce a strict weak ordering on the values.

\textbf{Template Parameters}
- \texttt{ExPolicy}: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it applies user-provided function objects.
- \texttt{Rng}: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- \texttt{Comp}: The type of the function/function object to use (deduced).
- \texttt{Proj}: The type of an optional projection function. This defaults to \texttt{util::projection_identity}

\textbf{Parameters}
- \texttt{policy}: The execution policy to use for the scheduling of the iterations.
- \texttt{rng}: Refers to the sequence of elements the algorithm will be applied to.
- \texttt{comp}: \texttt{comp} is a callable object. The return value of the \texttt{INVOKE} operation applied to an object of type \texttt{Comp}, when contextually converted to \texttt{bool}, yields true if the first argument of the call is less than the second, and false otherwise. It is assumed that \texttt{comp} will not apply any non-constant function through the dereferenced iterator.
- \texttt{proj}: Specifies the function (or function object) which will be invoked for each pair of elements as a projection operation before the actual predicate \texttt{comp} is invoked.

The application of function objects in parallel algorithm invoked with an execution policy object of type \texttt{sequenced_policy} execute in sequential order in the calling thread.

The application of function objects in parallel algorithm invoked with an execution policy object of type \texttt{parallel_policy} or \texttt{parallel_task_policy} are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

\textbf{Return} The \texttt{sort} algorithm returns a \texttt{hpx::future<Iter>} if the execution policy is of type \texttt{sequenced_task_policy} or \texttt{parallel_task_policy} and returns \texttt{Iter} otherwise. It returns \texttt{last}.

```cpp
#include <hpx/parallel/container_algorithms/count.hpp>

namespace hpx
{
    namespace parallel
    {
        namespace v1
        {
            \textbf{Functions}
            \begin{verbatim}
            template <typename ExPolicy, typename Rng, typename T, typename Proj = util::projection_identity>
            \end{verbatim}
        }
    }
}
```
Returns the number of elements in the range [first, last) satisfying a specific criteria. This version counts the elements that are equal to the given value.

The comparisons in the parallel count algorithm invoked with an execution policy object of type sequenced_policy execute in sequential order in the calling thread.

**Note** Complexity: Performs exactly last - first comparisons.

**Template Parameters**
- `ExPolicy`: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the comparisons.
- `Rng`: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- `T`: The type of the value to search for (deduced).
- `Proj`: The type of an optional projection function. This defaults to util::projection_identity

**Parameters**
- `policy`: The execution policy to use for the scheduling of the iterations.
- `rng`: Refers to the sequence of elements the algorithm will be applied to.
- `value`: The value to search for.
- `proj`: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate is invoked.

**Note** The comparisons in the parallel count algorithm invoked with an execution policy object of type parallel_policy or parallel_task_policy are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

**Return** The count algorithm returns a hpx::future<difference_type> if the execution policy is of type sequenced_task_policy or parallel_task_policy and returns difference_type otherwise (where difference_type is defined by std::iterator_traits<FwdIter>::difference_type). The count algorithm returns the number of elements satisfying the given criteria.

```
template<typename ExPolicy, typename Rng, typename F, typename Proj = util::projection_identity>
util::detail::algorithm_result<ExPolicy, typename std::iterator_traits<typename hpx::traits::range_traits<Rng>::iterator_type>::difference_type>::type
count_if(ExPolicy&& policy, Rng&& rng, F&& f, Proj&& proj = Proj())
```

Returns the number of elements in the range [first, last) satisfying a specific criteria. This version counts elements for which predicate `f` returns true.
**Note** Complexity: Performs exactly last - first applications of the predicate.

**Note** The assignments in the parallel count_if algorithm invoked with an execution policy object of type sequenced_policy execute in sequential order in the calling thread.

**Note** The assignments in the parallel count_if algorithm invoked with an execution policy object of type parallel_policy or parallel_task_policy are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

**Return** The count_if algorithm returns hpx::future<difference_type> if the execution policy is of type sequenced_task_policy or parallel_task_policy and returns difference_type otherwise (where difference_type is defined by std::iterator_traits<FwdIter>::difference_type. The count algorithm returns the number of elements satisfying the given criteria.

**Template Parameters**
- ExPolicy: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the comparisons.
- Rng: The type of the source range used (deduced). The iterators extracted from this range type must meet the requirements of an input iterator.
- F: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of count_if requires F to meet the requirements of CopyConstructible.
- Proj: The type of an optional projection function. This defaults to util::projection_identity

**Parameters**
- policy: The execution policy to use for the scheduling of the iterations.
- rng: Refers to the sequence of elements the algorithm will be applied to.
- f: Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by [first, last). This is an unary predicate which returns true for the required elements. The signature of this predicate should be equivalent to:

```cpp
bool pred(const Type &a);
```

The signature does not need to have const&, but the function must not modify the objects passed to it. The type Type must be such that an object of type FwdIter can be dereferenced and then implicitly converted to Type.
- proj: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate is invoked.

```cpp
#include <hpx/parallel/util/partitioner_with_cleanup.hpp>
#include <hpx/parallel/util/transfer.hpp>
namespace hpx

namespace parallel

namespace util

Functions

```cpp
template <typename InIter, typename OutIter>
std::pair<InIter, OutIter> copy (InIter first, InIter last, OutIter dest)

template <typename InIter, typename OutIter>
HPX_HOST_DEVICE std::pair<InIter, OutIter> hpx::parallel::util::copy_n(InIter first, std::size_t count, OutIter dest)

template <typename InIter, typename OutIter>
void copy_synchronize (InIter const &first, OutIter const &dest)
```
\begin{verbatim}

\textbf{template} <\texttt{typename InIter, typename OutIter}>
\texttt{std::pair<InIter, OutIter> move (InIter first, InIter last, OutIter dest)}

\textbf{template} <\texttt{typename InIter, typename OutIter}>
\texttt{std::pair<InIter, OutIter> move_n (InIter first, std::size_t count, OutIter dest)}

#include <hpx/parallel/util/scan_partitioner.hpp>
#include <hpx/parallel/util/projection_identity.hpp>

namespace hpx

namespace parallel

namespace util

\begin{verbatim}

struct projection_identity

\textbf{Public Functions}

\textbf{template} <\texttt{typename T}>
\texttt{HPX\_HOST\_DEVICE T\&\& hpx::parallel::util::projection_identity::operator() (T \&\& t)}

#include <hpx/parallel/util/invoke_projected.hpp>

namespace hpx

namespace parallel

namespace util

\begin{verbatim}

\textbf{template} <\texttt{typename Pred, typename Proj}>
\textbf{struct invoke_projected}

\textbf{Public Types}

\textbf{typedef} hpx::util::decay<Pred>::type \texttt{pred_type}
\textbf{typedef} hpx::util::decay<Proj>::type \texttt{proj_type}

\textbf{Public Functions}

\textbf{template} <\texttt{typename Pred\_, typename Proj\_}>
\textbf{invoke_projected (Pred\_ \&\&pred, Proj\_ \&\&proj)}

\textbf{template} <\texttt{typename T}>
\texttt{auto operator() (T \&\&t)}
\end{verbatim}
\end{verbatim}

\end{verbatim}
\end{verbatim}

Public Members

pred_type pred_
proj_type proj_

#include <hpx/parallel/util/cancellation_token.hpp>

namespace hpx

namespace parallel

namespace util

template <typename T = detail::no_data, typename Pred = std::less_equal<T>>
class cancellation_token

Public Functions

cancellation_token (T data)
bool was_cancelled (T data) const
void cancel (T data)
T get_data () const

Private Types

typedef std::atomic<T> flag_type

Private Members

std::shared_ptr<flag_type> was_cancelled_

#include <hpx/parallel/util/foreach_partitioner.hpp>
#include <hpx/parallel/util/loop.hpp>

namespace hpx

namespace parallel

namespace util

Functions

template <typename ExPolicy, typename VecOnly, typename F, typename... Iters>
HPX_HOST_DEVICE std::enable_if< !execution::is_vectorpack_execution_policy<ExPolicy>::value, typename hpx::util::invoke_result<F, Iters...>::type>::type hpx::parallel::util::loop_step(VecOnly, F && f, Iters &... its)

template <typename ExPolicy, typename Iter>
HPX_HOST_DEVICE std::enable_if< !execution::is_vectorpack_execution_policy<ExPolicy>::value, bool>::type hpx::parallel::util::loop_optimization(Iter, Iter)
template <typename ExPolicy, typename Begin, typename End, typename F>
HPX_HOST_DEVICE Begin hpx::parallel::util::loop(ExPolicy &&, Begin begin, End end, F && f)

template <typename ExPolicy, typename Begin, typename End, typename CancelToken, typename F>
HPX_HOST_DEVICE Begin hpx::parallel::util::loop(ExPolicy &&, Begin begin, End end, CancelToken && tok, F && f)

template <typename ExPolicy, typename VecOnly, typename Begin1, typename End1, typename Begin2, typename End2, typename F>
HPX_HOST_DEVICE std::enable_if< !execution::is_vectorpack_execution_policy<ExPolicy>::value, std::pair<Begin1, Begin2> >::type hpx::parallel::util::loop2(VecOnly, Begin1 begin1, End1 end1, Begin2 begin2, End2 end2, F && f)

#include <hpx/parallel/util/prefetching.hpp>

namespace hpx

namespace parallel

namespace util

2.9. API reference 743
Functions

\textbf{template} \textless \texttt{typename} Itr, \texttt{typename...} Ts\textgreater
\begin{verbatim}
\text{detail::prefetcher_context}<\text{Itr, Ts const...}>
\text{make_prefetcher_context}(Itr \text{ base_begin,
Itr \text{ base_end,}
std::size_t \text{ p_factor, Ts \text{ const}...\text{ rngs}})
\end{verbatim}

#include \textless \hpx/parallel/util/partitioner.hpp\textgreater
#include \textless \hpx/parallel/util/zip_iterator.hpp\textgreater
#include \textless \hpx/parallel/util/compare_projected.hpp\textgreater

namespace hpx

    namespace parallel

        namespace util

            \textbf{template} \textless \texttt{typename} Compare, \texttt{typename} Proj\textgreater
            \begin{verbatim}
            \text{template<>
            \text{struct compare_projected<Compare, Proj>}
            \end{verbatim}

            \textbf{Public Functions}

            \textbf{template} \textless \texttt{typename} Compare_, \texttt{typename} Proj_\textgreater
            \begin{verbatim}
            \text{compare_projected(Compare_ \&\&comp, Proj_ \&\&proj)}
            \end{verbatim}

            \textbf{template} \textless \texttt{typename} T1, \texttt{typename} T2\textgreater
            \begin{verbatim}
            \text{bool operator() (T1 \&\&t1, T2 \&\&t2) const}
            \end{verbatim}

            \textbf{Public Members}

            Compare \texttt{comp_}
            Proj \texttt{proj_}

            \textbf{template} \textless \texttt{typename} Compare, \texttt{typename} Proj1, \texttt{typename} Proj2\textgreater
            \begin{verbatim}
            \text{template<>
            \text{struct compare_projected<Compare, Proj1, Proj2>}
            \end{verbatim}

            \textbf{Public Functions}

            \textbf{template} \textless \texttt{typename} Compare_, \texttt{typename} Proj1_, \texttt{typename} Proj2_\textgreater
            \begin{verbatim}
            \text{compare_projected(Compare_ \&\&comp, Proj1_ \&\&proj1, Proj2_ \&\&proj2)}
            \end{verbatim}

            \textbf{template} \textless \texttt{typename} T1, \texttt{typename} T2\textgreater
            \begin{verbatim}
            \text{bool operator() (T1 \&\&t1, T2 \&\&t2) const}
            \end{verbatim}
Public Members

Compare `comp_
Proj1 `proj1_
Proj2 `proj2_

#include <hpx/parallel/util/transform_loop.hpp>

namespace hpx

namespace parallel

namespace util

Functions

`template <typename ExPolicy, typename Iter, typename OutIter, typename F>
HPX_HOST_DEVICE std::pair<Iter, OutIter> hpx::parallel::util::transform_loop(ExPolicy &&, Iter it, Iter end, OutIter dest, F && f)
` 

`template <typename ExPolicy, typename InIter1, typename InIter2, typename OutIter, typename F>
HPX_HOST_DEVICE std::enable_if< !execution::is_vectorpack_execution_policy<ExPolicy>::value, hpx::util::tuple<InIter1, ... >::type hpx::parallel::util::transform_binary_loop(InIter1 first1, InIter1 last1, InIter2 first2, OutIter dest, F && f)
` 

`template <typename ExPolicy, typename InIter1, typename InIter2, typename OutIter, typename F>
HPX_HOST_DEVICE std::enable_if< !execution::is_vectorpack_execution_policy<ExPolicy>::value, std::pair<Iter, OutIter> >::type hpx::parallel::util::transform_loop_n(Iter it, std::size_t count, OutIter dest, F && f)
` 

#include <hpx/parallel/algorithms/reduce.hpp>

namespace hpx

namespace parallel

namespace v1

Functions

`template <typename ExPolicy, typename FwdIterB, typename FwdIterE, typename T, typename F>
std::enable_if<execution::is_execution_policy<ExPolicy>::value, typename util::detail::algorithm_result<ExPolicy, T>::type>::type
hpx::parallel::v1::reduce(ExPolicy && policy, FwdIterB first, FwdIterE last, T init, F && f)
`
Returns GENERALIZED_SUM(f, init, *first, . . . , *(first + (last - first) - 1)).

The reduce operations in the parallel reduce algorithm invoked with an execution policy object of type sequenced_policy execute in sequential order in the calling thread.

**Note** Complexity: O(last - first) applications of the predicate f.

**Template Parameters**
- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- **FwdIterB**: The type of the source begin iterator used (deduced). This iterator type must meet the requirements of an forward iterator.
- **FwdIterE**: The type of the source end iterator used (deduced). This iterator type must meet the requirements of an forward iterator.
- **F**: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of copy_if requires F to meet the requirements of CopyConstructible.
- **T**: The type of the value to be used as initial (and intermediate) values (deduced).

**Parameters**
- **policy**: The execution policy to use for the scheduling of the iterations.
- **first**: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- **last**: Refers to the end of the sequence of elements the algorithm will be applied to.
- **f**: Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by [first, last). This is a binary predicate. The signature of this predicate should be equivalent to:

```cpp
Ret fun(const Type1 &a, const Type1 &b);
```

The signature does not need to have const&. The types Type1 Ret must be such that an object of type FwdIterB can be dereferenced and then implicitly converted to any of those types.
- **init**: The initial value for the generalized sum.

The reduce operations in the parallel copy_if algorithm invoked with an execution policy object of type parallel_policy or parallel_task_policy are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

The difference between reduce and accumulate is that the behavior of reduce may be non-deterministic for non-associative or non-commutative binary predicate.

**Return** The reduce algorithm returns a hpx::future<T> if the execution policy is of type sequenced_task_policy or parallel_task_policy and returns T otherwise. The reduce algorithm returns the result of the generalized sum over the elements given by the input range [first, last).

**Note** GENERALIZED_SUM(op, a1, . . . , aN) is defined as follows:
- a1 when N is 1
- op(GENERALIZED_SUM(op, b1, . . . , bK), GENERALIZED_SUM(op, bM, . . . , bN)), where:
  - b1, . . . , bN may be any permutation of a1, . . . , aN and
  - 1 < K+1 = M <= N.

**template** <typename ExPolicy, typename FwdIterB, typename FwdIterE, typename T>
std::enable_if<execution::is_execution_policy<ExPolicy>::value, typename util::detail::algorithm_result<ExPolicy, T>
Returns GENERALIZED_SUM(+, init, *first, . . . , *(first + (last - first) - 1)).

The reduce operations in the parallel reduce algorithm invoked with an execution policy object of type sequenced_policy execute in sequential order in the calling thread.

**Note** Complexity: \(O(last - first)\) applications of the operator+().

**Template Parameters**
- `ExPolicy`: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- `FwdIterB`: The type of the source begin iterator used (deduced). This iterator type must meet the requirements of an forward iterator.
- `FwdIterE`: The type of the source end iterator used (deduced). This iterator type must meet the requirements of an forward iterator.
- `T`: The type of the value to be used as initial (and intermediate) values (deduced).

**Parameters**
- `policy`: The execution policy to use for the scheduling of the iterations.
- `first`: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- `last`: Refers to the end of the sequence of elements the algorithm will be applied to.
- `init`: The initial value for the generalized sum.

The reduce operations in the parallel copy_if algorithm invoked with an execution policy object of type parallel_policy or parallel_task_policy are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

The difference between reduce and accumulate is that the behavior of reduce may be non-deterministic for non-associative or non-commutative binary predicate.

**Return** The reduce algorithm returns a `hpx::future<T>` if the execution policy is of type sequenced_task_policy or parallel_task_policy and returns `T` otherwise. The reduce algorithm returns the result of the generalized sum (applying operator+) over the elements given by the input range `[first, last)`.

**Note** GENERALIZED_SUM(+, a1, . . . , aN) is defined as follows:
- a1 when N is 1
- \(\text{opt}(\text{GENERALIZED_SUM}(+, b1, \ldots, bK), \text{GENERALIZED_SUM}(+, bM, \ldots, bN))\),
  where:
  - b1, . . . , bN may be any permutation of a1, . . . , aN and
  - 1 < K+1 = M <= N.

```template <typename ExPolicy, typename FwdIterB, typename FwdIterE>
std::enable_if<execution::is_execution_policy<ExPolicy>::value, typename util::detail::algorithm_result<ExPolicy, typename std::iterator_traits<FwdIterB>::value_type>::type>::type
reduce<ExPolicy, FwdIterB, FwdIterE>(ExPolicy&& policy, FwdIterB first, FwdIterE last)```

Returns GENERALIZED_SUM(+, T(), *first, . . . , *(first + (last - first) - 1)).

The reduce operations in the parallel reduce algorithm invoked with an execution policy object of type sequenced_policy execute in sequential order in the calling thread.

**Note** Complexity: \(O(last - first)\) applications of the operator+().

**Template Parameters**
- `ExPolicy`: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- `FwdIterB`: The type of the source begin iterator used (deduced). This iterator type must meet the requirements of an forward iterator.
FwdIterE: The type of the source end iterator used (deduced). This iterator type must meet the requirements of an forward iterator.

**Parameters**
- **policy**: The execution policy to use for the scheduling of the iterations.
- **first**: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- **last**: Refers to the end of the sequence of elements the algorithm will be applied to.

The reduce operations in the parallel copy_if algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

The difference between **reduce** and **accumulate** is that the behavior of reduce may be non-deterministic for non-associative or non-commutative binary predicate.

**Return** The **reduce** algorithm returns a `hpx::future<T>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns T otherwise (where T is the value_type of FwdIterB). The **reduce** algorithm returns the result of the generalized sum (applying operator+()) over the elements given by the input range [first, last).

**Note** The type of the initial value (and the result type) T is determined from the value_type of the used FwdIterB.

**Note** GENERALIZED_SUM(+, a1, . . . , aN) is defined as follows:
- a1 when N is 1
- op(GENERALIZED_SUM(+, b1, . . . , bK), GENERALIZED_SUM(+, bM, . . . , bN)), where:
  - b1, . . . , bN may be any permutation of a1, . . . , aN and
  - 1 < K+1 = M <= N.

```cpp
#include <hpx/parallel/algorithms/partition.hpp>

namespace hpx
{

namespace parallel
{

namespace v1
{

Functions

```cpp
template <typename ExPolicy, typename BidirIter, typename F, typename Proj = util::projection_identity>
util::detail::algorithm_result<ExPolicy, BidirIter>::type stable_partition(ExPolicy &&policy,
BidirIter first,
BidirIter last,
F &&f,
Proj &&proj = Proj())
{

Permutes the elements in the range [first, last) such that there exists an iterator i such that for every iterator j in the range [first, i) INVOKE(f, INVOKE (proj, *j)) != false, and for every iterator k in the range [i, last), INVOKE(f, INVOKE (proj, *k)) == false

The invocations of f in the parallel stable_partition algorithm invoked with an execution policy object of type `sequenced_policy` executes in sequential order in the calling thread.

**Note** Complexity: At most \( (last - first) \times \log(last - first) \) swaps, but only linear number of swaps if there is enough extra memory. Exactly `last - first` applications of the predicate and projection.

**Template Parameters**
• **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the invocations of $f$.

• **BidirIter**: The type of the source iterators used (deduced). This iterator type must meet the requirements of an input iterator.

• **F**: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of *transform* requires $F$ to meet the requirements of *CopyConstructible*.

• **Proj**: The type of an optional projection function. This defaults to util::projection_identity

### Parameters

- **policy**: The execution policy to use for the scheduling of the iterations.
- **first**: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- **last**: Refers to the end of the sequence of elements the algorithm will be applied to.
- **f**: Unary predicate which returns true if the element should be ordered before other elements. Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by [*first, last*). The signature of this predicate should be equivalent to:

```cpp
bool fun(const Type &a);
```

The signature does not need to have const&. The type *Type* must be such that an object of type *BidirIter* can be dereferenced and then implicitly converted to *Type*.

- **proj**: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate $f$ is invoked.

The invocations of $f$ in the parallel *stable_partition* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

### Return

The *stable_partition* algorithm returns an iterator $i$ such that for every iterator $j$ in the range [*first, i*), $f(*j) \neq \text{false}$ \text{and} $\text{INVOK}(f, \text{INVOK}(\text{proj}, *j)) \neq \text{false}$, and for every iterator $k$ in the range [*i, last*), $f(*k) \neq \text{false}$ \text{and} $\text{INVOK}(f, \text{INVOK}(\text{proj}, *k)) \neq \text{false}$. The relative order of the elements in both groups is preserved. If the execution policy is of type *parallel_task_policy* the algorithm returns a future<> referring to this iterator.

```cpp
template<typename ExPolicy, typename FwdIter, typename Pred, typename Proj = util::projection_identity>
util::detail::algorithm_result<ExPolicy, FwdIter>::type
partition(ExPolicy &&policy, FwdIter first, FwdIter last, Pred &&pred, Proj &&proj = Proj())
```

Reorders the elements in the range [*first, last*) in such a way that all elements for which the predicate *pred* returns true precede the elements for which the predicate *pred* returns false. Relative order of the elements is not preserved.

The assignments in the parallel *partition* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.

**Note** Complexity: At most $2 \times \text{(last - first)}$ swaps. Exactly *last - first* applications of the predicate and projection.

### Template Parameters

- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.

- **FwdIter**: The type of the source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.

- **Pred**: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of *partition* requires *Pred* to meet the requirements of *CopyConstructible*.

- **Proj**: The type of an optional projection function. This defaults to util::projection_identity
Parameters
- **policy**: The execution policy to use for the scheduling of the iterations.
- **first**: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- **last**: Refers to the end of the sequence of elements the algorithm will be applied to.
- **pred**: Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by [first, last). This is an unary predicate for partitioning the source iterators. The signature of this predicate should be equivalent to:

  ```cpp
  bool pred(const Type &a);
  ```

  The signature does not need to have const&, but the function must not modify the objects passed to it. The type Type must be such that an object of type InIter can be dereferenced and then implicitly converted to Type.
- **proj**: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate is invoked.

The assignments in the parallel partition algorithm invoked with an execution policy object of type parallel_policy or parallel_task_policy are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

**Return**

The partition algorithm returns a hpx::future<FwdIter> if the execution policy is of type parallel_task_policy and returns FwdIter otherwise. The partition algorithm returns the iterator to the first element of the second group.

```cpp
template<typename ExPolicy, typename FwdIter1, typename FwdIter2, typename FwdIter3, typename Pred, typename Proj = util::projection_identity>
util::detail::algorithm_result<ExPolicy, hpx::util::tagged_tuple<tag::in(FwdIter1), tag::out1 FwdIter2, tag::out2FwdIter3>>,::type partition_copy(ExPolicy &&policy, FwdIter1 first, FwdIter1 last, FwdIter2 dest_true, FwdIter3 dest_false, Pred &&pred, Proj &&proj = Proj())Copies the elements in the range, defined by [first, last), to two different ranges depending on the value returned by the predicate pred. The elements, that satisfy the predicate pred, are copied to the range beginning at dest_true. The rest of the elements are copied to the range beginning at dest_false. The order of the elements is preserved.
```

The assignments in the parallel partition_copy algorithm invoked with an execution policy object of type sequenced_policy execute in sequential order in the calling thread.

**Note**

Complexity: Performs not more than last - first assignments, exactly last - first applications of the predicate f.

**Template Parameters**
- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- **FwdIter1**: The type of the source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.
- **FwdIter2**: The type of the iterator representing the destination range for the elements that satisfy the predicate pred (deduced). This iterator type must meet the requirements of an forward iterator.
- **FwdIter3**: The type of the iterator representing the destination range for the elements that don’t satisfy the predicate pred (deduced). This iterator type must meet the requirements of an forward iterator.
- **Pred**: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of partition_copy requires Pred to meet the requirements of CopyConstructible.
- **Proj**: The type of an optional projection function. This defaults to util::projection_identity

**Parameters**
- **policy**: The execution policy to use for the scheduling of the iterations.
• **first**: Refers to the beginning of the sequence of elements the algorithm will be applied to.
• **last**: Refers to the end of the sequence of elements the algorithm will be applied to.
• **dest_true**: Refers to the beginning of the destination range for the elements that satisfy the predicate `pred`.
• **dest_false**: Refers to the beginning of the destination range for the elements that don’t satisfy the predicate `pred`.
• **pred**: Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by `[first, last)`. This is an unary predicate for partitioning the source iterators. The signature of this predicate should be equivalent to:

```cpp
bool pred(const Type &a);
```

The signature does not need to have const&, but the function must not modify the objects passed to it. The type `Type` must be such that an object of type `FwdIter1` can be dereferenced and then implicitly converted to `Type`.
• **proj**: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate is invoked.

The assignments in the parallel `partition_copy` algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

**Return**
The `partition_copy` algorithm returns a `hpx::future<tagged_tuple<tag::in(InIter), tag::out1(OutIter1), tag::out2(OutIter2)>>` if the execution policy is of type `parallel_task_policy` and returns `tagged_tuple<tag::in(InIter), tag::out1(OutIter1), tag::out2(OutIter2)>>` otherwise. The `partition_copy` algorithm returns the tuple of the source iterator `last`, the destination iterator to the end of the `dest_true` range, and the destination iterator to the end of the `dest_false` range.

```cpp
#include <hpx/parallel/algorithms/rotate.hpp>
namespace hpx
{
    namespace parallel
    {
        namespace v1
        {
            template<typename ExPolicy, typename FwdIter>
            util::detail::algorithm_result<ExPolicy, hpx::util::tagged_pair<tag::begin(FwdIter), tag::end(FwdIter)>>::type rotate(ExPolicy &&policy, FwdIter first, FwdIter new_first, FwdIter last);

            // Performs a left rotation on a range of elements. Specifically, rotate swaps the elements in the range [first, last) in such a way that the element new_first becomes the first element of the new range and new_first - 1 becomes the last element.
            template<typename ExPolicy, typename FwdIter>
            util::detail::algorithm_result<ExPolicy, hpx::util::tagged_pair<tag::begin(FwdIter), tag::end(FwdIter)>>::type rotate(ExPolicy &&policy, FwdIter first, FwdIter new_first, FwdIter last);
        }
    }
}
```

2.9. API reference
• FwdIter: The type of the source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.

Parameters
• policy: The execution policy to use for the scheduling of the iterations.
• first: Refers to the beginning of the sequence of elements the algorithm will be applied to.
• new_first: Refers to the element that should appear at the beginning of the rotated range.
• last: Refers to the end of the sequence of elements the algorithm will be applied to.

The assignments in the parallel rotate algorithm invoked with an execution policy object of type parallel_policy or parallel_task_policy are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Note The type of dereferenced FwdIter must meet the requirements of MoveAssignable and MoveConstructible.

Return The rotate algorithm returns a hpx::future<tagged_pair<tag::begin(FwdIter),
tag::end(FwdIter)>> > if the execution policy is of type parallel_task_policy and returns tagged_pair<tag::begin(FwdIter), tag::end(FwdIter)>> otherwise. The rotate algorithm returns the iterator equal to pair(first + (last - new_first), last).

template <typename ExPolicy, typename FwdIter1, typename FwdIter2>
util::detail::algorithm_result<ExPolicy, hpx::util::tagged_pair<tag::in(FwdIter1), tag::out FwdIter2>>::type
rotate_copy(ExPolicy &&policy, FwdIter1 first, FwdIter1 new_first, FwdIter1 last, FwdIter2 dest_first)Copies the elements from the range [first, last), to another range beginning at dest_first in such a way, that the element new_first becomes the first element of the new range and new_first - 1 becomes the last element.

The assignments in the parallel rotate_copy algorithm invoked with an execution policy object of type sequenced_policy execute in sequential order in the calling thread.

Note Complexity: Performs exactly last - first assignments.

Template Parameters
• ExPolicy: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
• FwdIter1: The type of the source iterators used (deduced). This iterator type must meet the requirements of an bidirectional iterator.
• FwdIter2: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an forward iterator.

Parameters
• policy: The execution policy to use for the scheduling of the iterations.
• first: Refers to the beginning of the sequence of elements the algorithm will be applied to.
• new_first: Refers to the element that should appear at the beginning of the rotated range.
• last: Refers to the end of the sequence of elements the algorithm will be applied to.
• dest_first: Refers to the begin of the destination range.

The assignments in the parallel rotate_copy algorithm invoked with an execution policy object of type parallel_policy or parallel_task_policy are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The rotate_copy algorithm returns a hpx::future<tagged_pair<tag::in(FwdIter1), tag::out(FwdIter2)>> > if the execution policy is of type parallel_task_policy and returns tagged_pair<tag::in(FwdIter1), tag::out(FwdIter2)>> otherwise. The rotate_copy algorithm returns the output iterator to the element past the last element copied.
#include <hpx/parallel/algorithms/all_any_none.hpp>

namespace hpx

namespace parallel

namespace v1

Functions

template <typename ExPolicy, typename FwdIter, typename F, typename Proj = util::projection_identity>
util::detail::algorithm_result<ExPolicy, bool>::type none_of(ExPolicy &&policy, FwdIter first, FwdIter last, F &&f, Proj &&proj = Proj())

Checks if unary predicate f returns true for no elements in the range [first, last).

The application of function objects in parallel algorithm invoked with an execution policy object of type sequenced_policy execute in sequential order in the calling thread.

Note Complexity: At most last - first applications of the predicate f

Template Parameters

- ExPolicy: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it applies user-provided function objects.
- FwdIter: The type of the source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.
- F: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of none_of requires F to meet the requirements of CopyConstructible.
- Proj: The type of an optional projection function. This defaults to util::projection_identity

Parameters

- policy: The execution policy to use for the scheduling of the iterations.
- first: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- last: Refers to the end of the sequence of elements the algorithm will be applied to.
- f: Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by [first, last). The signature of this predicate should be equivalent to:

  ```
  bool pred(const Type &a);
  ```

  The signature does not need to have const&, but the function must not modify the objects passed to it. The type Type must be such that an object of type FwdIter can be dereferenced and then implicitly converted to Type.
- proj: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate is invoked.

The application of function objects in parallel algorithm invoked with an execution policy object of type parallel_policy or parallel_task_policy are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The none_of algorithm returns a hpx::future<bool> if the execution policy is of type sequenced_task_policy or parallel_task_policy and returns bool otherwise. The none_of algorithm returns true if the unary predicate f returns true for no elements in the range, false otherwise. It returns true if the range is empty.
util::detail::algorithm_result<ExPolicy, bool>::type any_of(ExPolicy &&policy, FwdIter first, FwdIter last, F &&f, Proj &&proj = Proj())

Checks if unary predicate f returns true for at least one element in the range [first, last).

The application of function objects in parallel algorithm invoked with an execution policy object of type sequenced_policy execute in sequential order in the calling thread.

**Note** Complexity: At most last - first applications of the predicate f

**Template Parameters**
- ExPolicy: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it applies user-provided function objects.
- FwdIter: The type of the source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.
- F: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of any_of requires F to meet the requirements of CopyConstructible.
- Proj: The type of an optional projection function. This defaults to util::projection_identity

**Parameters**
- policy: The execution policy to use for the scheduling of the iterations.
- first: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- last: Refers to the end of the sequence of elements the algorithm will be applied to.
- f: Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by [first, last). The signature of this predicate should be equivalent to:

```cpp
bool pred(const Type &a);
```

The signature does not need to have const&, but the function must not modify the objects passed to it. The type Type must be such that an object of type FwdIter can be dereferenced and then implicitly converted to Type.
- proj: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate is invoked.

The application of function objects in parallel algorithm invoked with an execution policy object of type parallel_policy or parallel_task_policy are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

**Return**
The any_of algorithm returns a hpx::future<bool> if the execution policy is of type sequenced_task_policy or parallel_task_policy and returns bool otherwise. The any_of algorithm returns true if the unary predicate f returns true for at least one element in the range, false otherwise. It returns false if the range is empty.

```cpp
template <typename ExPolicy, typename FwdIter, typename F, typename Proj = util::projection_identity>
util::detail::algorithm_result<ExPolicy, bool>::type all_of(ExPolicy &&policy, FwdIter first, FwdIter last, F &&f, Proj &&proj = Proj())
```

Checks if unary predicate f returns true for all elements in the range [first, last).

The application of function objects in parallel algorithm invoked with an execution policy object of type sequenced_policy execute in sequential order in the calling thread.

**Note** Complexity: At most last - first applications of the predicate f

**Template Parameters**
- ExPolicy: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it applies user-provided function objects.
- FwdIter: The type of the source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.
• `F`: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of `all_of` requires `F` to meet the requirements of `CopyConstructible`.
• `Proj`: The type of an optional projection function. This defaults to `util::projection_identity`

**Parameters**

• `policy`: The execution policy to use for the scheduling of the iterations.
• `first`: Refers to the beginning of the sequence of elements the algorithm will be applied to.
• `last`: Refers to the end of the sequence of elements the algorithm will be applied to.
• `f`: Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by `[first, last)`. The signature of this predicate should be equivalent to:

```cpp
bool pred(const Type &a);
```

The signature does not need to have `const&`, but the function must not modify the objects passed to it. The type `Type` must be such that an object of type `FwdIter` can be dereferenced and then implicitly converted to `Type`.
• `proj`: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate is invoked.

The application of function objects in parallel algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

**Return** The `all_of` algorithm returns a `hpx::future<bool>` if the execution policy is of type `sequenced_policy` or `parallel_task_policy` and returns `bool` otherwise. The `all_of` algorithm returns true if the unary predicate `f` returns true for all elements in the range, false otherwise. It returns true if the range is empty.

```cpp
#include <hpx/parallel/algorithms/find.hpp>

namespace hpx
{
    namespace parallel
    {
        namespace v1
        {
            Functions
            template <typename ExPolicy, typename FwdIter, typename T>
            std::enable_if<execution::is_execution_policy<ExPolicy>::value, typename util::detail::algorithm_result<ExPolicy, FwdIter, T>::type>::type
            find(ExPolicy &&policy, FwdIter first, FwdIter last, T const &val)
            Returns the first element in the range `[first, last)` that is equal to `value`

            The comparison operations in the parallel `find` algorithm invoked with an execution policy object of type `sequenced_policy` execute in sequential order in the calling thread.
            **Note** Complexity: At most last - first applications of the operator==().
        }
    }
}
```
Template Parameters
- ExPolicy: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- FwdIter: The type of the source iterators used for the first range (deduced). This iterator type must meet the requirements of an forward iterator.
- T: The type of the value to find (deduced).

Parameters
- policy: The execution policy to use for the scheduling of the iterations.
- first: Refers to the beginning of the sequence of elements of the first range the algorithm will be applied to.
- last: Refers to the end of the sequence of elements of the first range the algorithm will be applied to.
- val: the value to compare the elements to

The comparison operations in the parallel find algorithm invoked with an execution policy object of type parallel_policy or parallel_task_policy are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The find algorithm returns a hpx::future<FwdIter> if the execution policy is of type sequenced_task_policy or parallel_task_policy and returns FwdIter otherwise. The find algorithm returns the first element in the range [first, last) that is equal to val. If no such element in the range of [first, last) is equal to val, then the algorithm returns last.

template <typename ExPolicy, typename FwdIter, typename F>
std::enable_if<execution::is_execution_policy<ExPolicy>::value, typename util::detail::algorithm_result<ExPolicy, FwdIter>::type>::type
find_if(ExPolicy &&policy, FwdIter first, FwdIter last, F &&f)

Returns the first element in the range [first, last) for which predicate f returns true

The comparison operations in the parallel find_if algorithm invoked with an execution policy object of type sequenced_policy execute in sequential order in the calling thread.

Note Complexity: At most last - first applications of the predicate.

Template Parameters
- ExPolicy: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- FwdIter: The type of the source iterators used for the first range (deduced). This iterator type must meet the requirements of a forward iterator.
- F: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of equal requires F to meet the requirements of CopyConstructible.

Parameters
- policy: The execution policy to use for the scheduling of the iterations.
- first: Refers to the beginning of the sequence of elements of the first range the algorithm will be applied to.
- last: Refers to the end of the sequence of elements of the first range the algorithm will be applied to.
- f: The unary predicate which returns true for the required element. The signature of the predicate should be equivalent to:
bool pred(const Type &a);

The signature does not need to have const &, but the function must not modify the objects passed to it. The type Type must be such that objects of type FwdIter can be dereferenced and then implicitly converted to Type.

The comparison operations in the parallel \texttt{find\_if} algorithm invoked with an execution policy object of type \texttt{parallel\_policy} or \texttt{parallel\_task\_policy} are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

\textbf{Return} The \texttt{find\_if} algorithm returns a \texttt{hpx::future\<FwdIter\>} if the execution policy is of type \texttt{sequenced\_task\_policy} or \texttt{parallel\_task\_policy} and returns \texttt{FwdIter} otherwise. The \texttt{find\_if} algorithm returns the first element in the range \texttt{[first,last)} that satisfies the predicate \(f\). If no such element exists that satisfies the predicate \(f\), the algorithm returns \texttt{last}.

\texttt{template <typename ExPolicy, typename FwdIter, typename F>
std::enable_if<execution::is_execution_policy<ExPolicy>::value, typename util::detail::algorithm_result<ExPolicy, FwdIter>::type>::type
find_if_not(ExPolicy &&policy, FwdIter first, FwdIter last, F &&f)}

Returns the first element in the range \texttt{[first, last)} for which predicate \(f\) returns false.

The comparison operations in the parallel \texttt{find\_if\_not} algorithm invoked with an execution policy object of type \texttt{sequenced\_policy} execute in sequential order in the calling thread.

\textbf{Note} Complexity: At most \texttt{last - first} applications of the predicate.

\textbf{Template Parameters}
- \texttt{ExPolicy}: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- \texttt{FwdIter}: The type of the source iterators used for the first range (deduced). This iterator type must meet the requirements of a forward iterator.
- \texttt{F}: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of \texttt{equal} requires \(F\) to meet the requirements of \texttt{CopyConstructible}.

\textbf{Parameters}
- \texttt{policy}: The execution policy to use for the scheduling of the iterations.
- \texttt{first}: Refers to the beginning of the sequence of elements of the first range the algorithm will be applied to.
- \texttt{last}: Refers to the end of the sequence of elements of the first range the algorithm will be applied to.
- \texttt{f}: The unary predicate which returns false for the required element. The signature of the predicate should be equivalent to:

\begin{verbatim}
bool pred(const Type &a);
\end{verbatim}

The signature does not need to have const &, but the function must not modify the objects passed to it. The type Type must be such that objects of type FwdIter can be dereferenced and then implicitly converted to Type.

The comparison operations in the parallel \texttt{find\_if\_not} algorithm invoked with an execution policy object of type \texttt{parallel\_policy} or \texttt{parallel\_task\_policy} are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

\textbf{Return} The \texttt{find\_if\_not} algorithm returns a \texttt{hpx::future\<FwdIter\>} if the execution policy is
of type `sequenced_task_policy` or `parallel_task_policy` and returns `FwdIter` otherwise. The `find_if_not` algorithm returns the first element in the range `[first, last)` that does **not** satisfy the predicate `f`. If no such element exists that does not satisfy the predicate `f`, the algorithm returns `last`.

```cpp
template <typename ExPolicy, typename FwdIter1, typename FwdIter2, typename Pred = detail::equal_to, typename Proj = util::projection_identity>
std::enable_if<execution::is_execution_policy<ExPolicy>::value, typename util::detail::algorithm_result<ExPolicy, FwdIter1>::type>::type
find_end(ExPolicy&& policy, FwdIter1 first1, FwdIter1 last1, FwdIter2 first2, FwdIter2 last2, Pred&& op = Pred(), Proj&& proj = Proj())
```

Returns the last subsequence of elements `[first2, last2)` found in the range `[first, last)` using the given predicate `f` to compare elements.

The comparison operations in the parallel `find_end` algorithm invoked with an execution policy object of type `sequenced_policy` execute in sequential order in the calling thread.

**Note** Complexity: at most `S*(N-S+1)` comparisons where `S = distance(first2, last2)` and `N = distance(first1, last1)`.

**Template Parameters**
- `ExPolicy`: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- `FwdIter1`: The type of the source iterators used for the first range (deduced). This iterator type must meet the requirements of an forward iterator.
- `FwdIter2`: The type of the source iterators used for the second range (deduced). This iterator type must meet the requirements of an forward iterator.
- `Pred`: The type of an optional function/function object to use. Unlike its sequential form, the parallel overload of `replace` requires `Pred` to meet the requirements of `CopyConstructible`. This defaults to `std::equal_to<>`
- `Proj`: The type of an optional projection function. This defaults to `util::projection_identity` and is applied to the elements of type dereferenced `FwdIter1` and dereferenced `FwdIter2`.

**Parameters**
- `policy`: The execution policy to use for the scheduling of the iterations.
- `first1`: Refers to the beginning of the sequence of elements of the first range the algorithm will be applied to.
- `last1`: Refers to the end of the sequence of elements of the first range the algorithm will be applied to.
- `first2`: Refers to the beginning of the sequence of elements the algorithm will be searching for.
- `last2`: Refers to the end of the sequence of elements of the algorithm will be searching for.
- `op`: The binary predicate which returns `true` if the elements should be treated as equal. The
signature should be equivalent to the following:

```cpp
bool pred(const Type1 &a, const Type2 &b);
```

The signature does not need to have const &, but the function must not modify the objects passed to it. The types Type1 and Type2 must be such that objects of types FwdIter1 and FwdIter2 can be dereferenced and then implicitly converted to Type1 and Type2 respectively.

- proj: Specifies the function (or function object) which will be invoked for each of the elements of type dereferenced FwdIter1 and dereferenced FwdIter2 as a projection operation before the function f is invoked.

The comparison operations in the parallel find_end algorithm invoked with an execution policy object of type parallel_policy or parallel_task_policy are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

This overload of find_end is available if the user decides to provide the algorithm their own predicate f.

**Return** The find_end algorithm returns a hpx::future<FwdIter> if the execution policy is of type sequenced_task_policy or parallel_task_policy and returns FwdIter otherwise. The find_end algorithm returns an iterator to the beginning of the last subsequence [first2, last2) in range [first, last). If the length of the subsequence [first2, last2) is greater than the length of the range [first1, last1), last1 is returned. Additionally if the size of the subsequence is empty or no subsequence is found, last1 is also returned.

**Template Parameters**

- ExPolicy: The type of the execution policy to use (deduced). It describes the manner in which

Searches the range [first, last) for any elements in the range [s_first, s_last). Uses binary predicate p to compare elements.

The comparison operations in the parallel find_first_of algorithm invoked with an execution policy object of type sequenced_policy execute in sequential order in the calling thread.

**Note** Complexity: at most ($S$*$N$) comparisons where $S$ = distance(s_first, s_last) and $N$ = distance(first, last).

**Template Parameters**

- ExPolicy: The type of the execution policy to use (deduced). It describes the manner in
which the execution of the algorithm may be parallelized and the manner in which it executes
the assignments.
• FwdIter1: The type of the source iterators used for the first range (deduced). This iterator
type must meet the requirements of an forward iterator.
• FwdIter2: The type of the source iterators used for the second range (deduced). This
iterator type must meet the requirements of an forward iterator.
• Pred: The type of an optional function/function object to use. Unlike its sequential form,
the parallel overload of equal requires Pred to meet the requirements of CopyConstructible.
This defaults to std::equal_to<>.
• Proj1: The type of an optional projection function. This defaults to
util::projection_identity and is applied to the elements of type dereferenced FwdIter1.
• Proj2: The type of an optional projection function. This defaults to
util::projection_identity and is applied to the elements of type dereferenced FwdIter2.

Parameters
• policy: The execution policy to use for the scheduling of the iterations.
• first: Refers to the beginning of the sequence of elements of the first range the algorithm
will be applied to.
• last: Refers to the end of the sequence of elements of the first range the algorithm will be
applied to.
• s_first: Refers to the beginning of the sequence of elements the algorithm will be search-
ing for.
• s_last: Refers to the end of the sequence of elements of the algorithm will be searching
for.
• op: The binary predicate which returns true if the elements should be treated as equal. The
signature should be equivalent to the following:

```cpp
bool pred(const Type1 &a, const Type2 &b);
```

The signature does not need to have const &, but the function must not modify the objects
passed to it. The types Type1 and Type2 must be such that objects of types FwdIter1 and
FwdIter2 can be dereferenced and then implicitly converted to Type1 and Type2 respectively.
• proj1: Specifies the function (or function object) which will be invoked for each of the
elements of type dereferenced FwdIter1 as a projection operation before the function op is
invoked.
• proj2: Specifies the function (or function object) which will be invoked for each of the
elements of type dereferenced FwdIter2 as a projection operation before the function op is
invoked.

The comparison operations in the parallel find_first_of algorithm invoked with an execution pol-
icy object of type parallel_policy or parallel_task_policy are permitted to execute in an unordered
fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The find_first_of algorithm returns a hpx::future<FwdIter1> if the execution policy is
of type sequenced_task_policy or parallel_task_policy and returns FwdIter1 otherwise. The
find_first_of algorithm returns an iterator to the first element in the range [first, last) that is
equal to an element from the range [s_first, s_last). If the length of the subsequence [s_first,
s_last) is greater than the length of the range [first, last), last is returned. Additionally if the size
of the subsequence is empty or no subsequence is found, last is also returned. This overload of
find_end is available if the user decides to provide the algorithm their own predicate f.

#include <hpx/parallel/algorithms/for_each.hpp>

namespace hpx

namespace parallel
namespace vl

Functions

```
template <typename ExPolicy, typename FwdIter, typename Size, typename F, typename Proj = util::projection_identity>
util::detail::algorithm_result<ExPolicy, FwdIter>::type for_each_n(
  ExPolicy &&policy,
  FwdIter first, Size count,
  F &&f, Proj &&proj = Proj())
```

Applies f to the result of dereferencing every iterator in the range [first, first + count), starting from first and proceeding to first + count - 1.

If f returns a result, the result is ignored.

Note Complexity: Applies f exactly count times.

If the type of first satisfies the requirements of a mutable iterator, f may apply non-constant functions through the dereferenced iterator.

Unlike its sequential form, the parallel overload of for_each does not return a copy of its Function parameter, since parallelization may not permit efficient state accumulation.

The application of function objects in parallel algorithm invoked with an execution policy object of type sequenced_policy execute in sequential order in the calling thread.

Template Parameters

- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it applies user-provided function objects.
- **FwdIter**: The type of the source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.
- **Size**: The type of the argument specifying the number of elements to apply f to.
- **F**: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of for_each requires F to meet the requirements of CopyConstructible.
- **Proj**: The type of an optional projection function. This defaults to util::projection_identity

Parameters

- **policy**: The execution policy to use for the scheduling of the iterations.
- **first**: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- **count**: Refers to the number of elements starting at first the algorithm will be applied to.
- **f**: Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by [first, last). The signature of this predicate should be equivalent to:

  ```
  <ignored> pred(const Type &a);
  ```

  The signature does not need to have const&. The type Type must be such that an object of type FwdIter can be dereferenced and then implicitly converted to Type.
- **proj**: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate f is invoked.

The application of function objects in parallel algorithm invoked with an execution policy object of type parallel_policy or parallel_task_policy are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The for_each_n algorithm returns a hpx::future<FwdIter> if the execution policy is of type sequenced_task_policy or parallel_task_policy and returns FwdIter otherwise. It returns first + count for non-negative values of count and first for negative values.
template <typename ExPolicy, typename FwdIter, typename F, typename Proj = util::projection_identity>
util::detail::algorithm_result<ExPolicy, FwdIter>::type
for_each(
    ExPolicy &&policy,
    FwdIter first, FwdIter last,
    F &&f, Proj &&proj = Proj() )

Applies f to the result of dereferencing every iterator in the range [first, last).
If f returns a result, the result is ignored.

Note Complexity: Applies f exactly last - first times.
If the type of first satisfies the requirements of a mutable iterator, f may apply non-constant
functions through the dereferenced iterator.

Unlike its sequential form, the parallel overload of for_each does not return a copy of its Function
parameter, since parallelization may not permit efficient state accumulation.

The application of function objects in parallel algorithm invoked with an execution policy object
of type sequenced_policy execute in sequential order in the calling thread.

Template Parameters
- ExPolicy: The type of the execution policy to use (deduced). It describes the manner in
  which the execution of the algorithm may be parallelized and the manner in which it applies
  user-provided function objects.
- FwdIter: The type of the source iterators used (deduced). This iterator type must meet the
  requirements of an forward iterator.
- F: The type of the function/function object to use (deduced). Unlike its sequential form, the
  parallel overload of for_each requires F to meet the requirements of CopyConstructible.
- Proj: The type of an optional projection function. This defaults to util::projection_identity

Parameters
- policy: The execution policy to use for the scheduling of the iterations.
- first: Refers to the beginning of the sequence of elements the algorithm will be applied
to.
- last: Refers to the end of the sequence of elements the algorithm will be applied to.
- f: Specifies the function (or function object) which will be invoked for each of the elements
  in the sequence specified by [first, last). The signature of this predicate should be equivalent
to:

```
<ignored> pred(const Type &a);
```

The signature does not need to have const&. The type Type must be such that an object of
type FwdIter can be dereferenced and then implicitly converted to Type.
- proj: Specifies the function (or function object) which will be invoked for each of the
  elements as a projection operation before the actual predicate f is invoked.

The application of function objects in parallel algorithm invoked with an execution policy object
of type parallel_policy or parallel_task_policy are permitted to execute in an unordered fashion
in unspecified threads, and indeterminately sequenced within each thread.

Return The for_each algorithm returns a hpx::future<FwdIter> if the execution policy is of type
sequenced_task_policy or parallel_task_policy and returns FwdIter otherwise. It returns last.

#include <hpx/parallel/algorithms/reverse.hpp>

namespace hpx

namespace parallel

namespace vl
Functions

```cpp
template <typename ExPolicy, typename BidirIter>
util::detail::algorithm_result<ExPolicy, BidirIter>::type reverse
(ExPolicy &&policy, BidirIter first, BidirIter last)
```

Reverses the order of the elements in the range [first, last). Behaves as if applying std::iter_swap to every pair of iterators first+i, (last-i) - 1 for each non-negative i < (last-first)/2.

The assignments in the parallel reverse algorithm invoked with an execution policy object of type sequenced_policy execute in sequential order in the calling thread.

**Note** Complexity: Linear in the distance between first and last.

**Template Parameters**

- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- **BidirIter**: The type of the source iterators used (deduced). This iterator type must meet the requirements of a bidirectional iterator.

**Parameters**

- **policy**: The execution policy to use for the scheduling of the iterations.
- **first**: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- **last**: Refers to the end of the sequence of elements the algorithm will be applied to.

The assignments in the parallel reverse algorithm invoked with an execution policy object of type parallel_policy or parallel_task_policy are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

**Return** The reverse algorithm returns a hpx::future<BidirIter> if the execution policy is of type sequenced_task_policy or parallel_task_policy and returns BidirIter otherwise. It returns last.

```cpp
template <typename ExPolicy, typename BidirIter, typename FwdIter>
util::detail::algorithm_result<ExPolicy, hpx::util::tagged_pair<tag::in(BidirIter), tag::out FwdIter>>::type reverse_copy
ExPolicy &&policy, BidirIter first, BidirIter last, FwdIter dest_first
```

Copies the elements from the range [first, last) to another range beginning at dest_first in such a way that the elements in the new range are in reverse order. Behaves as if by executing the assignment *(dest_first + (last - first) - 1 - i) = *(first + i) once for each non-negative i < (last - first) If the source and destination ranges (that is, [first, last) and [dest_first, dest_first+(last-first)) respectively) overlap, the behavior is undefined.

The assignments in the parallel reverse_copy algorithm invoked with an execution policy object of type sequenced_policy execute in sequential order in the calling thread.

**Note** Complexity: Performs exactly last - first assignments.

**Template Parameters**

- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- **BidirIter**: The type of the source iterators used (deduced). This iterator type must meet the requirements of a bidirectional iterator.
- **FwdIter**: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of a forward iterator.

**Parameters**

- **policy**: The execution policy to use for the scheduling of the iterations.
- **first**: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- **last**: Refers to the end of the sequence of elements the algorithm will be applied to.
- **dest_first**: Refers to the begin of the destination range.
The assignments in the parallel reverse_copy algorithm invoked with an execution policy object of type parallel_policy or parallel_task_policy are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

**Return** The reverse_copy algorithm returns a hpx::future<tagged_pair<tag::in(BidirIter), tag::out(FwdIter)>> if the execution policy is of type sequenced_task_policy or parallel_task_policy and returns tagged_pair<tag::in(BidirIter), tag::out(FwdIter)>> otherwise. The copy algorithm returns the pair of the input iterator forwarded to the first element after the last in the input sequence and the output iterator to the element in the destination range, one past the last element copied.

```
#include <hpx/parallel/algorithms/is_partitioned.hpp>

namespace hpx

namespace parallel

namespace v1

Functions

template <typename ExPolicy, typename FwdIter, typename Pred>
std::enable_if<execution::is_execution_policy<ExPolicy>::value, typename util::detail::algorithm_result<ExPolicy, bool>::type>::type
is_partitioned(ExPolicy &&policy, FwdIter first, FwdIter last, Pred &&pred)
```

Determines if the range (first, last) is partitioned.

The predicate operations in the parallel is_partitioned algorithm invoked with an execution policy object of type sequenced_policy executes in sequential order in the calling thread.

**Note** Complexity: at most (N) predicate evaluations where \( N = \text{distance}(\text{first}, \text{last}) \).

**Template Parameters**
- ExPolicy: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- FwdIter: The type of the source iterators used for the This iterator type must meet the requirements of a forward iterator.

**Parameters**
- policy: The execution policy to use for the scheduling of the iterations.
- first: Refers to the beginning of the sequence of elements of that the algorithm will be applied to.
- last: Refers to the end of the sequence of elements of that the algorithm will be applied to.
- pred: Refers to the binary predicate which returns true if the first argument should be treated as less than the second argument. The signature of the function should be equivalent to

```cpp
bool pred(const Type &a, const Type &b);
```
The signature does not need to have const &, but the function must not modify the objects passed to it. The type Type must be such that objects of types FwdIter can be dereferenced and then implicitly converted to Type.

The comparison operations in the parallel is_partitioned algorithm invoked with an execution policy object of type parallel_policy or parallel_task_policy are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

**Return** The is_partitioned algorithm returns a hpx::future<bool> if the execution policy is of type task_execution_policy and returns bool otherwise. The is_partitioned algorithm returns true if each element in the sequence for which pred returns true precedes those for which pred returns false. Otherwise is_partitioned returns false. If the range [first, last) contains less than two elements, the function is always true.

```cpp
#include <hpx/parallel/algorithms/transform_reduce_binary.hpp>

namespace hpx
{
    namespace parallel
    {
        namespace v1
        {
            template <typename ExPolicy, typename FwdIter1, typename FwdIter2, typename T>
            util::detail::algorithm_result<ExPolicy, T>::type transform_reduce(ExPolicy &&policy, FwdIter1 first1, FwdIter1 last1, FwdIter2 first2, T init)
            {
                // Implementation...
            }
        }
    }
}
```

Returns the result of accumulating init with the inner products of the pairs formed by the elements of two ranges starting at first1 and first2.

The operations in the parallel transform_reduce algorithm invoked with an execution policy object of type sequenced_policy execute in sequential order in the calling thread.

**Note** Complexity: O(last - first) applications of the predicate op2.

**Template Parameters**
- ExPolicy: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- FwdIter1: The type of the first source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.
- FwdIter2: The type of the second source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.
- T: The type of the value to be used as return) values (deduced).

**Parameters**
- policy: The execution policy to use for the scheduling of the iterations.
- first1: Refers to the beginning of the first sequence of elements the result will be calculated with.
- last1: Refers to the end of the first sequence of elements the algorithm will be applied to.
- first2: Refers to the beginning of the second sequence of elements the result will be calculated with.
- init: The initial value for the sum.

The operations in the parallel transform_reduce algorithm invoked with an execution policy object of type parallel_policy or parallel_task_policy are permitted to execute in an unordered fashion...
in unspecified threads, and indeterminately sequenced within each thread.

**Return** The *transform_reduce* algorithm returns a `hpx::future<T>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `T` otherwise.

```cpp
template <typename ExPolicy, typename FwdIter1, typename FwdIter2, typename T, typename Reduce, typename ConvOp>
util::detail::algorithm_result<ExPolicy, T>::type transform_reduce(ExPolicy &&policy, FwdIter1 first1, FwdIter1 last1, FwdIter2 first2, T init, Reduce &&red_op, ConvOp &&conv_op)
```

Returns the result of accumulating `init` with the inner products of the pairs formed by the elements of two ranges starting at `first1` and `first2`.

The operations in the parallel *transform_reduce* algorithm invoked with an execution policy object of type `sequenced_policy` execute in sequential order in the calling thread.

**Note** Complexity: O(`last - first`) applications of the predicate `op2`.

**Template Parameters**
- `ExPolicy`: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- `FwdIter1`: The type of the first source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.
- `FwdIter2`: The type of the second source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.
- `T`: The type of the value to be used as return values (deduced).
- `Reduce`: The type of the binary function object used for the multiplication operation.
- `ConvOp`: The type of the unary function object used to transform the elements of the input sequence before invoking the reduce function.

**Parameters**
- `policy`: The execution policy to use for the scheduling of the iterations.
- `first1`: Refers to the beginning of the first sequence of elements the result will be calculated with.
- `last1`: Refers to the end of the first sequence of elements the algorithm will be applied to.
- `first2`: Refers to the beginning of the second sequence of elements the result will be calculated with.
- `init`: The initial value for the sum.
- `red_op`: Specifies the function (or function object) which will be invoked for the initial value and each of the return values of `op2`. This is a binary predicate. The signature of this predicate should be equivalent to should be equivalent to:

```cpp
Ret fun(const Type1 &a, const Type1 &b);
```

The signature does not need to have `const&`, but the function must not modify the objects passed to it. The type `Ret` must be such that it can be implicitly converted to a type of `T`.
- `conv_op`: Specifies the function (or function object) which will be invoked for each of the input values of the sequence. This is a binary predicate. The signature of this predicate should be equivalent to:

```cpp
Ret fun(const Type1 &a, const Type2 &b);
```

The signature does not need to have `const&`, but the function must not modify the objects passed to it. The type `Ret` must be such that it can be implicitly converted to an object for the second argument type of `op1`.

The operations in the parallel *transform_reduce* algorithm invoked with an execution policy object
of type parallel_policy or parallel_task_policy are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

**Return** The transform_reduce algorithm returns a hpx::future\(<T>\) if the execution policy is of type sequenced_task_policy or parallel_task_policy and returns \(T\) otherwise.

```cpp
#include <hpx/parallel/algorithms/uninitialized_value_construct.hpp>

namespace hpx
{
    namespace parallel
    {
        namespace v1
        {

            Functions

            template <typename ExPolicy, typename FwdIter>
                util::detail::algorithm_result<ExPolicy>::type
                    uninitialized_value_construct(ExPolicy &&policy, FwdIter first, FwdIter last)

            Constructs objects of type typename iterator_traits<ForwardIt>::value_type in the uninitialized storage designated by the range [first, last) by default-initialization. If an exception is thrown during the initialization, the function has no effects.

            The assignments in the parallel uninitialized_value_construct algorithm invoked with an execution policy object of type sequenced_policy execute in sequential order in the calling thread.

            Note Complexity: Performs exactly \(last - first\) assignments.

            Template Parameters
            • ExPolicy: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
            • FwdIter: The type of the source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.

            Parameters
            • policy: The execution policy to use for the scheduling of the iterations.
            • first: Refers to the beginning of the sequence of elements the algorithm will be applied to.
            • last: Refers to the end of the sequence of elements the algorithm will be applied to.

            The assignments in the parallel uninitialized_value_construct algorithm invoked with an execution policy object of type parallel_policy or parallel_task_policy are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

            Return The uninitialized_value_construct algorithm returns a hpx::future\(<void>\), if the execution policy is of type sequenced_task_policy or parallel_task_policy and returns \(void\) otherwise.

            template <typename ExPolicy, typename FwdIter, typename Size>
```
util::detail::algorithm_result<ExPolicy, FwdIter>::type uninitialized_value_construct_n (ExPolicy &&policy, FwdIter first, Size count)

Constructs objects of type typename iterator_traits<ForwardIt>::value_type in the uninitialized storage designated by the range [first, first + count) by default-initialization. If an exception is thrown during the initialization, the function has no effects.

The assignments in the parallel uninitialized_value_construct_n algorithm invoked with an execution policy object of type sequenced_policy execute in sequential order in the calling thread.

**Note** Complexity: Performs exactly count assignments, if count > 0, no assignments otherwise.

**Template Parameters**
- ExPolicy: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- FwdIter: The type of the source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.
- Size: The type of the argument specifying the number of elements to apply f to.

**Parameters**
- policy: The execution policy to use for the scheduling of the iterations.
- first: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- count: Refers to the number of elements starting at first the algorithm will be applied to.

The assignments in the parallel uninitialized_value_construct_n algorithm invoked with an execution policy object of type parallel_policy or parallel_task_policy are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

**Return** The uninitialized_value_construct_n algorithm returns a hpx::future<FwdIter> if the execution policy is of type sequenced_task_policy or parallel_task_policy and returns FwdIter otherwise. The uninitialized_value_construct_n algorithm returns the iterator to the element in the source range, one past the last element constructed.

```cpp
#include <hpx/parallel/algorithms/uninitialized_move.hpp>

namespace hpx
{

namespace parallel
{

namespace v1
{

Functions

template <typename ExPolicy, typename FwdIter1, typename FwdIter2>
util::detail::algorithm_result<ExPolicy, FwdIter2>::type uninitialized_move (ExPolicy &&policy, FwdIter1 first, FwdIter1 last, FwdIter2 dest)
```
Moves the elements in the range, defined by \([\text{first}, \text{last})\), to an uninitialized memory area beginning at \(\text{dest}\). If an exception is thrown during the initialization, some objects in \([\text{first}, \text{last})\) are left in a valid but unspecified state.

The assignments in the parallel \textit{uninitialized\_move} algorithm invoked with an execution policy object of type \textit{sequenced\_policy} execute in sequential order in the calling thread.

\textbf{Note} Complexity: Performs exactly \textit{last - first} move operations.

\textbf{Template Parameters}

- \textit{ExPolicy}: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- \textit{FwdIter1}: The type of the source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.
- \textit{FwdIter2}: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of a forward iterator.

\textbf{Parameters}

- \textit{policy}: The execution policy to use for the scheduling of the iterations.
- \textit{first}: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- \textit{last}: Refers to the end of the sequence of elements the algorithm will be applied to.
- \textit{dest}: Refers to the beginning of the destination range.

The assignments in the parallel \textit{uninitialized\_move} algorithm invoked with an execution policy object of type \textit{parallel\_policy} or \textit{parallel\_task\_policy} are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

\textbf{Return} The \textit{uninitialized\_move} algorithm returns a \textit{hpx::future<FwdIter2>}, if the execution policy is of type \textit{sequenced\_task\_policy} or \textit{parallel\_task\_policy} and returns \textit{FwdIter2} otherwise. The \textit{uninitialized\_move} algorithm returns the output iterator to the element in the destination range, one past the last element moved.

\textbf{Example}

\begin{verbatim}
#include <hpx/hpx.hpp>
#include <hpx/util/uninitialized_move.hpp>

int main() {
    using namespace hpx;
    using namespace hpx::util;
    using namespace hpx::util::tagged_pair;

    int arr1[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
    int arr2[10];

    // Use sequenced policy
    sequenced_policy policy;
    hpx::future<hpx::util::tagged_pair<tag::in, FwdIter1>, tag::out, FwdIter2>> result = uninitialized_move(sequenced_policy, arr1, arr2);
    // Use parallel policy
    parallel_policy policy;
    hpx::future<hpx::util::tagged_pair<tag::in, FwdIter1>, tag::out, FwdIter2>> result = uninitialized_move(parallel_policy, arr1, arr2);

    // Use parallel task policy
    parallel_task_policy policy;
    hpx::future<hpx::util::tagged_pair<tag::in, FwdIter1>, tag::out, FwdIter2>> result = uninitialized_move(parallel_task_policy, arr1, arr2);
}
\end{verbatim}
The assignments in the parallel `uninitialized_move_n` algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

**Return** The `uninitialized_move_n` algorithm returns a `hpx::future<std::pair<FwdIter1, FwdIter2>>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `std::pair<FwdIter1, FwdIter2>` otherwise. The `uninitialized Move_n` algorithm returns the pair of the input iterator to the element past in the source range and an output iterator to the element in the destination range, one past the last element moved.

```cpp
#include <hpx/parallel/algorithms/search.hpp>

namespace hpx
{
namespace parallel
{
namespace v1
{

Functions

```template<typename ExPolicy, typename FwdIter, typename FwdIter2, typename Pred = detail::equal_to,
util::detail::algorithm_result<ExPolicy, FwdIter>::type
search(ExPolicy&& policy, FwdIter first, FwdIter last, FwdIter2 s_first, FwdIter2 s_last,
Pred&& op = Pred(), Proj1&& proj1 = Proj1(), Proj2&& proj2 = Proj2())```

Searches the range `[first, last)` for any elements in the range `[s_first, s_last)`. Uses a provided predicate to compare elements.

The comparison operations in the parallel `search` algorithm invoked with an execution policy object of type `sequenced_policy` execute in sequential order in the calling thread.

**Note** Complexity: at most \((S*N)\) comparisons where \(S = \text{distance(s_first, s_last)}\) and \(N = \text{distance(first, last)}\).

**Template Parameters**
- `ExPolicy`: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- `FwdIter`: The type of the source iterators used for the first range (deduced). This iterator type must meet the requirements of an input iterator.
- `FwdIter2`: The type of the source iterators used for the second range (deduced). This iterator type must meet the requirements of an forward iterator.
- `Proj1`: The type of an optional projection function. This defaults to `util::projection_identity` and is applied to the elements of type dereferenced `FwdIter`.
- `Proj2`: The type of an optional projection function. This defaults to `util::projection_identity` and is applied to the elements of type dereferenced `FwdIter2`.

**Parameters**
- `policy`: The execution policy to use for the scheduling of the iterations.
- `first`: Refers to the beginning of the sequence of elements of the first range the algorithm will be applied to.
• last: Refers to the end of the sequence of elements of the first range the algorithm will be applied to.
• s_first: Refers to the beginning of the sequence of elements the algorithm will be searching for.
• s_last: Refers to the end of the sequence of elements of the algorithm will be searching for.
• op: Refers to the binary predicate which returns true if the elements should be treated as equal. The signature of the function should be equivalent to

```cpp
bool pred(const Type1 &a, const Type2 &b);
```

The signature does not need to have const &, but the function must not modify the objects passed to it. The types Type1 and Type2 must be such that objects of types FwdIter1 and FwdIter2 can be dereferenced and then implicitly converted to Type1 and Type2 respectively.

• proj1: Specifies the function (or function object) which will be invoked for each of the elements of type dereferenced FwdIter1 as a projection operation before the actual predicate is invoked.
• proj2: Specifies the function (or function object) which will be invoked for each of the elements of type dereferenced FwdIter2 as a projection operation before the actual predicate is invoked.

The comparison operations in the parallel search algorithm invoked with an execution policy object of type parallel_policy or parallel_task_policy are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The search algorithm returns a hpx: :future< FwdIter > if the execution policy is of type task_execution_policy and returns FwdIter otherwise. The search algorithm returns an iterator to the beginning of the first subsequence [s_first, s_last) in range [first, last). If the length of the subsequence [s_first, s_last) is greater than the length of the range [first, last), last is returned. Additionally if the size of the subsequence is empty first is returned. If no subsequence is found, last is returned.

```cpp
template <typename ExPolicy, typename FwdIter, typename FwdIter2, typename Pred = detail::equal_to,
          typename Proj1 = util::projection_identity,
          typename Proj2 = util::projection_identity>
util::detail::algorithm_result<ExPolicy, FwdIter>::type
search_n(ExPolicy && policy, FwdIter first, std::size_t count, FwdIter2 s_first, FwdIter2 s_last, Pred && op = Pred(), Proj1 && proj1 = Proj1(), Proj2 && proj2 = Proj2())
```

Searches the range [first, last) for any elements in the range [s_first, s_last). Uses a provided predicate to compare elements.

The comparison operations in the parallel search_n algorithm invoked with an execution policy object of type sequenced_policy execute in sequential order in the calling thread.

Note Complexity: at most (S*N) comparisons where S = distance(s_first, s_last) and N = count.

Template Parameters
- ExPolicy: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- FwdIter: The type of the source iterators used for the first range (deduced). This iterator type must meet the requirements of an input iterator.
- FwdIter2: The type of the source iterators used for the second range (deduced). This iterator type must meet the requirements of an forward iterator.
- Pred: The type of an optional function/function object to use. Unlike its sequential form, the parallel overload of adjacent_find requires Pred to meet the requirements of CopyConstructible. This defaults to std::equal_to<>
Parameters

- **policy**: The execution policy to use for the scheduling of the iterations.
- **first**: Refers to the beginning of the sequence of elements of the first range the algorithm will be applied to.
- **count**: Refers to the range of elements of the first range the algorithm will be applied to.
- **s_first**: Refers to the beginning of the sequence of elements the algorithm will be searching for.
- **s_last**: Refers to the end of the sequence of elements the algorithm will be searching for.
- **op**: Refers to the binary predicate which returns true if the elements should be treated as equal. The signature of the function should be equivalent to

  ```cpp
  bool pred(const Type1 &a, const Type2 &b);
  ```

  The signature does not need to have const &, but the function must not modify the objects passed to it. The types `Type1` and `Type2` must be such that objects of types `FwdIter1` and `FwdIter2` can be dereferenced and then implicitly converted to `Type1` and `Type2` respectively.
- **proj1**: Specifies the function (or function object) which will be invoked for each of the elements of type dereferenced `FwdIter1` as a projection operation before the actual predicate is invoked.
- **proj2**: Specifies the function (or function object) which will be invoked for each of the elements of type dereferenced `FwdIter2` as a projection operation before the actual predicate is invoked.

The comparison operations in the parallel `search_n` algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

**Return**: The `search_n` algorithm returns a `hpx::future<FwdIter>` if the execution policy is of type `task_execution_policy` and returns `FwdIter` otherwise. The `search_n` algorithm returns an iterator to the beginning of the last subsequence `[s_first, s_last)` in range `[first, first+count)`. If the length of the subsequence `[s_first, s_last)` is greater than the length of the range `[first, first+count)`, `first` is returned. Additionally if the size of the subsequence is empty or no subsequence is found, `first` is also returned.

```cpp
#include <hpx/parallel/algorithms/is_sorted.hpp>

namespace hpx
{
    namespace parallel
    {
        namespace v1
        {
            template <typename ExPolicy, typename FwdIter, typename Pred = detail::less>
            Functions
        }
    }
}
```
std::enable_if<execution::is_execution_policy<ExPolicy>::value, typename util::detail::algorithm_result<ExPolicy, bool>::type>::type is_sorted(ExPolicy&& policy, FwdIter first, FwdIter last, Pred&& pred = Pred())

Determines if the range [first, last) is sorted. Uses pred to compare elements.

The comparison operations in the parallel is_sorted algorithm invoked with an execution policy object of type sequenced_policy executes in sequential order in the calling thread.

**Note** Complexity: at most (N+S-1) comparisons where N = distance(first, last). S = number of partitions

**Template Parameters**
- ExPolicy: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- FwdIter: The type of the source iterators used for the This iterator type must meet the requirements of a forward iterator.
- Pred: The type of an optional function/function object to use. Unlike its sequential form, the parallel overload of is_sorted requires Pred to meet the requirements of CopyConstructible. This defaults to std::less<>

**Parameters**
- policy: The execution policy to use for the scheduling of the iterations.
- first: Refers to the beginning of the sequence of elements of that the algorithm will be applied to.
- last: Refers to the end of the sequence of elements of that the algorithm will be applied to.
- pred: Refers to the binary predicate which returns true if the first argument should be treated as less than the second argument. The signature of the function should be equivalent to

```cpp
bool pred(const Type &a, const Type &b);
```

The signature does not need to have const &, but the function must not modify the objects passed to it. The type Type must be such that objects of types FwdIter can be dereferenced and then implicitly converted to Type.

The comparison operations in the parallel is_sorted algorithm invoked with an execution policy object of type parallel_policy or parallel_task_policy are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

**Return** The is_sorted algorithm returns a hpx::future<bool> if the execution policy is of type task_execution_policy and returns bool otherwise. The is_sorted algorithm returns a bool if each element in the sequence [first, last) satisfies the predicate passed. If the range [first, last) contains less than two elements, the function always returns true.

```cpp
template <typename ExPolicy, typename FwdIter, typename Pred = std::less>
```
std::enable_if<execution::is_execution_policy<ExPolicy>::value, typename util::detail::algorithm_result<ExPolicy, FwdIter>::type>::type

Returns the first element in the range [first, last) that is not sorted. Uses a predicate to compare elements or the less than operator.

The comparison operations in the parallel is_sorted_until algorithm invoked with an execution policy object of type sequenced_policy executes in sequential order in the calling thread.

**Note** Complexity: at most (N+S-1) comparisons where N = distance(first, last). S = number of partitions

**Template Parameters**
- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- **FwdIter**: The type of the source iterators used for the This iterator type must meet the requirements of a forward iterator.
- **Pred**: The type of an optional function/function object to use. Unlike its sequential form, the parallel overload of is_sorted_until requires Pred to meet the requirements of CopyConstructible. This defaults to std::less<>.

**Parameters**
- **policy**: The execution policy to use for the scheduling of the iterations.
- **first**: Refers to the beginning of the sequence of elements of that the algorithm will be applied to.
- **last**: Refers to the end of the sequence of elements of that the algorithm will be applied to.
- **pred**: Refers to the binary predicate which returns true if the first argument should be treated as less than the second argument. The signature of the function should be equivalent to

```cpp
bool pred(const Type &a, const Type &b);
```

The signature does not need to have const &., but the function must not modify the objects passed to it. The type **Type** must be such that objects of types FwdIter can be dereferenced and then implicitly converted to Type.

The comparison operations in the parallel is_sorted_until algorithm invoked with an execution policy object of type parallel_policy or parallel_task_policy are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

**Return** The is_sorted_until algorithm returns a hpx::future<FwdIter> if the execution policy is of type task_execution_policy and returns FwdIter otherwise. The is_sorted_until algorithm returns the first unsorted element. If the sequence has less than two elements or the sequence is sorted, last is returned.

#include <hpx/parallel/algorithms/reduce_by_key.hpp>

namespace hpx

namespace parallel
namespace v1

Functions

template <typename ExPolicy, typename RanIter, typename RanIter2, typename FwdIter1, typename FwdIter2, typename Compare = std::equal_to<typename std::iterator_traits<RanIter>::value_type>, typename Func = std::plus<typename std::iterator_traits<RanIter2>::value_type>>
util::detail::algorithm_result<ExPolicy, std::pair<FwdIter1, FwdIter2>>::type reduce_by_key(ExPolicy &&policy, RanIter key_first, RanIter key_last, RanIter2 values_first, FwdIter1 keys_output, FwdIter2 values_output, Compare &&comp = Compare(), Func &&func = Func())

Reduce by Key performs an inclusive scan reduction operation on elements supplied in key/value pairs. The algorithm produces a single output value for each set of equal consecutive keys in [key_first, key_last). The value being the GENERALIZED_NONCOMMUTATIVE_SUM(op, init, *first, . . . , *(first + (i - result))). for the run of consecutive matching keys. The number of keys supplied must match the number of values.

comp has to induce a strict weak ordering on the values.

Note Complexity: $O(last - first)$ applications of the predicate op.

Template Parameters

- ExPolicy: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it applies user-provided function objects.
- RanIter: The type of the key iterators used (deduced). This iterator type must meet the requirements of a random access iterator.
- RanIter2: The type of the value iterators used (deduced). This iterator type must meet the requirements of a random access iterator.
- FwdIter1: The type of the iterator representing the destination key range (deduced). This iterator type must meet the requirements of an forward iterator.
- FwdIter2: The type of the iterator representing the destination value range (deduced). This iterator type must meet the requirements of an forward iterator.
• **Compare**: The type of the optional function/function object to use to compare keys (deduced). Assumed to be `std::equal_to` otherwise.

• **Func**: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of `copy_if` requires `F` to meet the requirements of `CopyConstructible`.

**Parameters**

• **policy**: The execution policy to use for the scheduling of the iterations.

• **key_first**: Refers to the beginning of the sequence of key elements the algorithm will be applied to.

• **key_last**: Refers to the end of the sequence of key elements the algorithm will be applied to.

• **values_first**: Refers to the beginning of the sequence of value elements the algorithm will be applied to.

• **keys_output**: Refers to the start output location for the keys produced by the algorithm.

• **values_output**: Refers to the start output location for the values produced by the algorithm.

• **comp**: `comp` is a callable object. The return value of the INVOKE operation applied to an object of type `Comp`, when contextually converted to `bool`, yields true if the first argument of the call is less than the second, and false otherwise. It is assumed that `comp` will not apply any non-constant function through the dereferenced iterator.

• **func**: Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by `[first, last)`. This is a binary predicate. The signature of this predicate should be equivalent to:

```cpp
Ret fun(const Type1 &a, const Type1 &b);
```

The signature does not need to have `const&`. The types `Type1 Ret` must be such that an object of type `FwdIter` can be dereferenced and then implicitly converted to any of those types.

The application of function objects in parallel algorithm invoked with an execution policy object of type `sequenced_policy` execute in sequential order in the calling thread.

The application of function objects in parallel algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

**Return** The `reduce_by_key` algorithm returns a `hpx::future<pair<Iter1,Iter2>>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `pair<Iter1,Iter2>` otherwise.

```cpp
#include <hpx/parallel/algorithms/fill.hpp>

namespace hpx
{

namespace parallel
{

namespace v1
{

Functions

```cpp
template <typename ExPolicy, typename FwdIter, typename T>
t util::detail::algorithm_result<ExPolicy>::type fill (ExPolicy &&policy, FwdIter first, FwdIter last, T value)
```

Assigns the given value to the elements in the range `[first, last)`. The comparisons in the parallel `fill` algorithm invoked with an execution policy object of type `sequenced_policy` execute in sequential order in the calling thread.
Complexity: Performs exactly last - first assignments.

**Template Parameters**

- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- **FwdIter**: The type of the source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.
- **T**: The type of the value to be assigned (deduced).

**Parameters**

- **policy**: The execution policy to use for the scheduling of the iterations.
- **first**: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- **last**: Refers to the end of the sequence of elements the algorithm will be applied to.
- **value**: The value to be assigned.

The comparisons in the parallel fill algorithm invoked with an execution policy object of type parallel_policy or parallel_task_policy are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

**Return**
The fill algorithm returns a hpx::future<void> if the execution policy is of type sequenced_task_policy or parallel_task_policy and returns difference_type otherwise (where difference_type is defined by void).

```cpp
template <typename ExPolicy, typename FwdIter, typename Size, typename T>
util::detail::algorithm_result<ExPolicy, FwdIter>::type fill_n (ExPolicy && policy, FwdIter first, Size count, T value)
```

Assigns the given value value to the first count elements in the range beginning at first if count > 0. Does nothing otherwise.

The comparisons in the parallel fill_n algorithm invoked with an execution policy object of type sequenced_policy execute in sequential order in the calling thread.

**Note** Complexity: Performs exactly count assignments, for count > 0.

**Template Parameters**

- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- **FwdIter**: The type of the source iterators used (deduced). This iterator type must meet the requirements of an output iterator.
- **Size**: The type of the argument specifying the number of elements to apply f to.
- **T**: The type of the value to be assigned (deduced).

**Parameters**

- **policy**: The execution policy to use for the scheduling of the iterations.
- **first**: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- **count**: Refers to the number of elements starting at first the algorithm will be applied to.
- **value**: The value to be assigned.

The comparisons in the parallel fill_n algorithm invoked with an execution policy object of type parallel_policy or parallel_task_policy are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

**Return**
The fill_n algorithm returns a hpx::future<void> if the execution policy is of type sequenced_task_policy or parallel_task_policy and returns difference_type otherwise (where difference_type is defined by void).
```cpp
#include <hpx/parallel/algorithms/transform_inclusive_scan.hpp>

namespace hpx
{
    namespace parallel
    {
        namespace v1
        {
            template <typename ExPolicy, typename FwdIter1, typename FwdIter2, typename Op, typename Conv, typename T>
            util::detail::algorithm_result<ExPolicy, FwdIter2>::type transform_inclusive_scan(ExPolicy &&policy, FwdIter1 first, FwdIter1 last, FwdIter2 dest, Op &&op, Conv &&conv, T init)
            Assigns through each iterator i in [result, result + (last - first)) the value of GENERALIZED_NONCOMMUTATIVE_SUM(op, init, conv(*first), ..., conv(*(first + (i - result)))).

            The reduce operations in the parallel transform_inclusive_scan algorithm invoked with an execution policy object of type sequenced_policy execute in sequential order in the calling thread.

            Note  Complexity: O(last - first) applications of the predicate op.

            Template Parameters
            • ExPolicy: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
            • FwdIter1: The type of the source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.
            • FwdIter2: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an forward iterator.
            • Conv: The type of the unary function object used for the conversion operation.
            • T: The type of the value to be used as initial (and intermediate) values (deduced).
            • Op: The type of the binary function object used for the reduction operation.

            Parameters
            • policy: The execution policy to use for the scheduling of the iterations.
            • first: Refers to the beginning of the sequence of elements the algorithm will be applied to.
            • last: Refers to the end of the sequence of elements the algorithm will be applied to.
            • dest: Refers to the beginning of the destination range.
            • conv: Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by [first, last). This is a unary predicate. The signature of this predicate should be equivalent to:
```

Chapter 2. What’s so special about HPX?
The signature does not need to have const&, but the function must not modify the objects passed to it. The type Type must be such that an object of type FwdIter1 can be dereferenced and then implicitly converted to Type. The type R must be such that an object of this type can be implicitly converted to T.

- **init**: The initial value for the generalized sum.
- **op**: Specifies the function (or function object) which will be invoked for each of the values of the input sequence. This is a binary predicate. The signature of this predicate should be equivalent to:

\[
\text{Ret fun(\text{const Type \&a, const Type \&b);}
\]

The signature does not need to have const&, but the function must not modify the objects passed to it. The types Type1 and Ret must be such that an object of a type as given by the input sequence can be implicitly converted to any of those types.

The reduce operations in the parallel transform_inclusive_scan algorithm invoked with an execution policy object of type parallel_policy or parallel_task_policy are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Neither conv nor op shall invalidate iterators or subranges, or modify elements in the ranges [first, last) or [result, result + (last - first)).

**Return** The copy_n algorithm returns a hpx::future<FwdIter2> if the execution policy is of type sequenced_task_policy or parallel_task_policy and returns FwdIter2 otherwise. The transform_inclusive_scan algorithm returns the output iterator to the element in the destination range, one past the last element copied.

**Note** GENERALIZED_NONCOMMUTATIVE_SUM(op, a1, ..., aN) is defined as:
- a1 when N is 1
- op(GENERALIZED_NONCOMMUTATIVE_SUM(op, a1, ..., aK), GENERALIZED_NONCOMMUTATIVE_SUM(op, aM, ..., aN)) where 1 < K+1 = M <= N.

The difference between exclusive_scan and transform_inclusive_scan is that transform_inclusive_scan includes the ith input element in the ith sum. If op is not mathematically associative, the behavior of transform_inclusive_scan may be non-deterministic.

```
template<typename ExPolicy, typename FwdIter1, typename FwdIter2, typename Conv, typename Op>
util::detail::algorithm_result<ExPolicy, FwdIter2>::type
transform_inclusive_scan(ExPolicy &&policy,
                        FwdIter1 first,
                        FwdIter1 last,
                        FwdIter2 dest,
                        Op &&op,
                        Conv &&conv)
```

Assigns through each iterator i in [result, result + (last - first)) the value of GENERALIZED_NONCOMMUTATIVE_SUM(op, conv(*first), ..., conv(*((first + (i - result))))).

The reduce operations in the parallel transform_inclusive_scan algorithm invoked with an execution policy object of type sequenced_policy execute in sequential order in the calling thread.

**Note** Complexity: O(last - first) applications of the predicate op.

**Template Parameters**
• **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
• **FwdIter1**: The type of the source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.
• **FwdIter2**: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an forward iterator.
• **Conv**: The type of the unary function object used for the conversion operation.
• **T**: The type of the value to be used as initial (and intermediate) values (deduced).
• **Op**: The type of the binary function object used for the reduction operation.

**Parameters**

- **policy**: The execution policy to use for the scheduling of the iterations.
- **first**: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- **last**: Refers to the end of the sequence of elements the algorithm will be applied to.
- **dest**: Refers to the beginning of the destination range.
- **conv**: Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by [first, last). This is a unary predicate. The signature of this predicate should be equivalent to:

```cpp
R fun(const Type &a);
```

The signature does not need to have const&, but the function must not modify the objects passed to it. The type `Type` must be such that an object of type `FwdIter1` can be dereferenced and then implicitly converted to `Type`. The type `R` must be such that an object of this type can be implicitly converted to `T`.

- **op**: Specifies the function (or function object) which will be invoked for each of the values of the input sequence. This is a binary predicate. The signature of this predicate should be equivalent to:

```cpp
Ret fun(const Type1 &a, const Type1 &b);
```

The signature does not need to have const&, but the function must not modify the objects passed to it. The types `Type1` and `Ret` must be such that an object of a type as given by the input sequence can be implicitly converted to any of those types.

The reduce operations in the parallel `transform_inclusive_scan` algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Neither `conv` nor `op` shall invalidate iterators or subranges, or modify elements in the ranges `[first, last)` or `[result, result + (last - first))].

**Return**

The `copy_n` algorithm returns a `hpx::future<FwdIter2>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `FwdIter2` otherwise. The `transform_inclusive_scan` algorithm returns the output iterator to the element in the destination range, one past the last element copied.

**Note**

`GENERALIZED_NONCOMMUTATIVE_SUM(op, a1, . . . , aN)` is defined as:

- `a1` when `N` is 1
- `op(GENERALIZED_NONCOMMUTATIVE_SUM(op, a1, . . . , aK), GENERALIZED_NONCOMMUTATIVE_SUM(op, aM, . . . , aN))` where `1 < K+1 = M <= N`.

The difference between `exclusive_scan` and `transform_inclusive_scan` is that `transform_inclusive_scan` includes the ith input element in the ith sum.
#include <hpx/parallel/algorithms/adjacent_find.hpp>

namespace hpx

namespace parallel

namespace v1

Functions

template <typename ExPolicy, typename FwdIter, typename Pred = detail::equal_to>
std::enable_if<
execution::is_execution_policy<ExPolicy>::value,
typename util::detail::algorithm_result<ExPolicy, FwdIter>::type>
adjacent_find(
ExPolicy &&policy,
FwdIter first,
FwdIter last,
Pred &&op = Pred())

Searches the range [first, last) for two consecutive identical elements. This version uses the given binary predicate op.

The comparison operations in the parallel adjacent_find invoked with an execution policy object of type sequenced_policy execute in sequential order in the calling thread.

Note Complexity: Exactly the smaller of (result - first) + 1 and (last - first) - 1 application of the predicate where result is the value returned.

Template Parameters

- ExPolicy: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- FwdIter: The type of the source iterators used for the range (deduced). This iterator type must meet the requirements of an forward iterator.
- Pred: The type of an optional function/function object to use. Unlike its sequential form, the parallel overload of adjacent_find requires Pred to meet the requirements of CopyConstructible. This defaults to std::equal_to<>.

Parameters

- policy: The execution policy to use for the scheduling of the iterations.
- first: Refers to the beginning of the sequence of elements of the range the algorithm will be applied to.
- last: Refers to the end of the sequence of elements of the range the algorithm will be applied to.
- op: The binary predicate which returns true if the elements should be treated as equal. The signature should be equivalent to the following:

```cpp
bool pred(const Typel &a, const Typel &b);
```

The signature does not need to have const &, but the function must not modify the objects passed to it. The types Typel must be such that objects of type FwdIter can be dereferenced and then implicitly converted to Typel.

The comparison operations in the parallel adjacent_find invoked with an execution policy object of type parallel_policy or parallel_task_policy are permitted to execute in an unordered fashion.
This overload of `adjacent_find` is available if the user decides to provide their algorithm their own binary predicate `op`.

**Return** The `adjacent_find` algorithm returns a `hpx::future<InIter>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `InIter` otherwise. The `adjacent_find` algorithm returns an iterator to the first of the identical elements. If no such elements are found, `last` is returned.

```cpp
#include <hpx/parallel/algorithms/set_intersection.hpp>

namespace hpx

namespace parallel

namespace v1

Functions

template <typename ExPolicy, typename FwdIter1, typename FwdIter2, typename FwdIter3, typename Pred = detail::less>
std::enable_if<execution::is_execution_policy<ExPolicy>::value, typename util::detail::algorithm_result<ExPolicy, FwdIter3>::type>::type
set_intersection(
    ExPolicy &&policy,
    FwdIter1 first1, FwdIter1 last1,
    FwdIter2 first2, FwdIter2 last2,
    FwdIter3 dest,
    Pred &&op = Pred())
```

Constructs a sorted range beginning at `dest` consisting of all elements present in both sorted ranges `[first1, last1)` and `[first2, last2)`. This algorithm expects both input ranges to be sorted with the given binary predicate `f`.

If some element is found `m` times in `[first1, last1)` and `n` times in `[first2, last2)`, the first `std::min(m, n)` elements will be copied from the first range to the destination range. The order of equivalent elements is preserved. The resulting range cannot overlap with either of the input ranges.

**Note** Complexity: At most `2*(N1 + N2 - 1)` comparisons, where `N1` is the length of the first sequence and `N2` is the length of the second sequence. The resulting range cannot overlap with either of the input ranges.

The application of function objects in parallel algorithm invoked with a sequential execution policy object execute in sequential order in the calling thread (`sequenced_policy`) or in a single new thread spawned from the current thread (`parallel_task_policy`).

**Template Parameters**
- `ExPolicy`: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it applies...
user-provided function objects.

- **FwdIter1**: The type of the source iterators used (deduced) representing the first sequence. This iterator type must meet the requirements of an forward iterator.
- **FwdIter2**: The type of the source iterators used (deduced) representing the first sequence. This iterator type must meet the requirements of an forward iterator.
- **FwdIter3**: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an output iterator.
- **Pred**: The type of an optional function/function object to use. Unlike its sequential form, the parallel overload of `set_intersection` requires `Pred` to meet the requirements of `CopyConstructible`. This defaults to `std::less<>`

### Parameters

- **policy**: The execution policy to use for the scheduling of the iterations.
- **first1**: Refers to the beginning of the sequence of elements of the first range the algorithm will be applied to.
- **last1**: Refers to the end of the sequence of elements of the first range the algorithm will be applied to.
- **first2**: Refers to the beginning of the sequence of elements of the second range the algorithm will be applied to.
- **last2**: Refers to the end of the sequence of elements of the second range the algorithm will be applied to.
- **dest**: Refers to the beginning of the destination range.
- **op**: The binary predicate which returns true if the elements should be treated as equal. The signature of the predicate function should be equivalent to the following:

```cpp
bool pred(const Type1 &a, const Type1 &b);
```

The signature does not need to have `const &`, but the function must not modify the objects passed to it. The type `Type1` must be such that objects of type `InIter` can be dereferenced and then implicitly converted to `Type1`.

The application of function objects in parallel algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

### Return

The `set_intersection` algorithm returns a `hpx::future<FwdIter3>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `FwdIter3` otherwise. The `set_intersection` algorithm returns the output iterator to the element in the destination range, one past the last element copied.

```cpp
#include <hpx/parallel/algorithms/sort_by_key.hpp>

namespace hpx

namespace parallel

namespace v1

Functions

```cpp
template<typename ExPolicy, typename KeyIter, typename ValueIter, typename Compare = detail::less>
util::detail::algorithm_result<ExPolicy, hpx::util::tagged_pair<tag::in1(KeyIter), tag::in2(ValueIter)>>::type sort_by_key(ExPolicy &&policy, KeyIter key_first, KeyIter key_last, ValueIter value_first, Compare &&comp = Compare())
``` Sorts one range of data using keys supplied in another range. The key elements in the range `[key_first, key_last)` are sorted in ascending order.
with the corresponding elements in the value range moved to follow the sorted order. The algorithm is not stable, the order of equal elements is not guaranteed to be preserved. The function uses the given comparison function object comp (defaults to using operator<()).

A sequence is sorted with respect to a comparator comp and a projection proj if for every iterator i pointing to the sequence and every non-negative integer n such that i + n is a valid iterator pointing to an element of the sequence, and INVOKE(comp, INVOKE(proj, *(i + n)), INVOKE(proj, *i)) == false.

**Note** Complexity: O(Nlog(N)), where N = std::distance(first, last) comparisons. comp has to induce a strict weak ordering on the values.

**Template Parameters**

- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it applies user-provided function objects.
- **KeyIter**: The type of the key iterators used (deduced). This iterator type must meet the requirements of a random access iterator.
- **ValueIter**: The type of the value iterators used (deduced). This iterator type must meet the requirements of a random access iterator.
- **Comp**: The type of the function/function object to use (deduced).

**Parameters**

- **policy**: The execution policy to use for the scheduling of the iterations.
- **key_first**: Refers to the beginning of the sequence of key elements the algorithm will be applied to.
- **key_last**: Refers to the end of the sequence of key elements the algorithm will be applied to.
- **value_first**: Refers to the beginning of the sequence of value elements the algorithm will be applied to, the range of elements must match [key_first, key_last]
- **comp**: comp is a callable object. The return value of the INVOKE operation applied to an object of type Comp, when contextually converted to bool, yields true if the first argument of the call is less than the second, and false otherwise. It is assumed that comp will not apply any non-constant function through the dereferenced iterator.

The application of function objects in parallel algorithm invoked with an execution policy object of type sequenced_policy execute in sequential order in the calling thread.

The application of function objects in parallel algorithm invoked with an execution policy object of type parallel_policy or parallel_task_policy are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

**Return** The sort_by-key algorithm returns a hpx:future<tagged_pair<tag::in1(KeyIter), tag::in2(ValueIter)>> if the execution policy is of type sequenced_task_policy or parallel_task_policy and returns otherwise. The algorithm returns a pair holding an iterator pointing to the first element after the last element in the input key sequence and an iterator pointing to the first element after the last element in the input value sequence.

```cpp
#include <hpx/parallel/algorithms/swap_ranges.hpp>

namespace hpx

namespace parallel

namespace v1
```
Functions

```cpp
template <typename ExPolicy, typename FwdIter1, typename FwdIter2>
std::enable_if<execution::is_execution_policy<ExPolicy>::value, typename util::detail::algorithm_result<ExPolicy, FwdIter2>::type>::type
swap_ranges(
    ExPolicy&& policy,
    FwdIter1 first1,
    FwdIter1 last1,
    FwdIter2 first2)
```

Exchanges elements between range [first1, last1) and another range starting at first2.

The swap operations in the parallel `swap_ranges` algorithm invoked with an execution policy object of type `sequenced_policy` execute in sequential order in the calling thread.

**Note** Complexity: Linear in the distance between `first1` and `last1`

**Template Parameters**
- `ExPolicy`: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the swap operations.
- `FwdIter1`: The type of the first range of iterators to swap (deduced). This iterator type must meet the requirements of an forward iterator.
- `FwdIter2`: The type of the second range of iterators to swap (deduced). This iterator type must meet the requirements of an forward iterator.

**Parameters**
- `policy`: The execution policy to use for the scheduling of the iterations.
- `first1`: Refers to the beginning of the first sequence of elements the algorithm will be applied to.
- `last1`: Refers to the end of the first sequence of elements the algorithm will be applied to.
- `first2`: Refers to the beginning of the second sequence of elements the algorithm will be applied to.

The swap operations in the parallel `swap_ranges` algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

**Return** The `swap_ranges` algorithm returns a `hpx::future<FwdIter2>` if the execution policy is of type `parallel_task_policy` and returns `FwdIter2` otherwise. The `swap_ranges` algorithm returns iterator to the element past the last element exchanged in the range beginning with `first2`.

```cpp
#include <hpx/parallel/algorithms/uninitialized_default_construct.hpp>

namespace hpx
{
    namespace parallel
    {
        namespace v1
        {
            Functions
            
            template <typename ExPolicy, typename FwdIter>
```

# 2.9. API reference
Constructs objects of type typename iterator_traits<ForwardIt>::value_type in the uninitialized storage designated by the range [first, last) by default-initialization. If an exception is thrown during the initialization, the function has no effects.

The assignments in the parallel uninitialized_default_construct algorithm invoked with an execution policy object of type sequenced_policy execute in sequential order in the calling thread.

Note Complexity: Performs exactly last - first assignments.

Template Parameters
- ExPolicy: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- FwdIter: The type of the source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.

Parameters
- policy: The execution policy to use for the scheduling of the iterations.
- first: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- last: Refers to the end of the sequence of elements the algorithm will be applied to.

The assignments in the parallel uninitialized_default_construct algorithm invoked with an execution policy object of type parallel_policy or parallel_task_policy are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The uninitialized_default_construct algorithm returns a hpx::future<void>, if the execution policy is of type sequenced_task_policy or parallel_task_policy and returns void otherwise.

template <typename ExPolicy, typename FwdIter, typename Size>
util::detail::algorithm_result<ExPolicy, FwdIter>::type uninitialized_default_construct_n(ExPolicy &policy, FwdIter first, FwdIter last, Size count)

Constructs objects of type typename iterator_traits<ForwardIt>::value_type in the uninitialized storage designated by the range [first, first + count) by default-initialization. If an exception is thrown during the initialization, the function has no effects.

The assignments in the parallel uninitialized_default_construct_n algorithm invoked with an execution policy object of type sequenced_policy execute in sequential order in the calling thread.

Note Complexity: Performs exactly count assignments, if count > 0, no assignments otherwise.

Template Parameters
- ExPolicy: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- FwdIter: The type of the source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.
- Size: The type of the argument specifying the number of elements to apply f to.

Parameters
• **policy**: The execution policy to use for the scheduling of the iterations.

• **first**: Refers to the beginning of the sequence of elements the algorithm will be applied to.

• **count**: Refers to the number of elements starting at `first` the algorithm will be applied to.

The assignments in the parallel `uninitialized_default_construct_n` algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

**Return** The `uninitialized_default_construct_n` algorithm returns a `hpx::future<FwdIter>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `FwdIter` otherwise. The `uninitialized_default_construct_n` algorithm returns the iterator to the element in the source range, one past the last element constructed.

```
#include <hpx/parallel/algorithms/merge.hpp>

namespace hpx

namespace parallel

namespace v1

Functions

```

```

template <typename ExPolicy, typename RandIter1, typename RandIter2, typename RandIter3, typename Comp = detail::less, typename Proj1 = util::projection_identity, typename Proj2 = util::projection_identity>

util::detail::algorithm_result<ExPolicy, hpx::util::tagged_tuple<tag::in1(RandIter1), tag::in2(RandIter2), tag::outRandIter3>> merge(ExPolicy &&policy, RandIter1 first1, RandIter1 last1, RandIter2 first2, RandIter2 last2, RandIter3 dest, Comp &&comp = Comp(), Proj1 &&proj1 = Proj1(), Proj2 &&proj2 = Proj2()) Merges two sorted ranges [first1, last1) and [first2, last2) into one sorted range beginning at dest. The order of equivalent elements in the each of original two ranges is preserved. For equivalent elements in the original two ranges, the elements from the first range precede the elements from the second range. The destination range cannot overlap with either of the input ranges.

The assignments in the parallel `merge` algorithm invoked with an execution policy object of type `sequenced_policy` execute in sequential order in the calling thread.

**Note** Complexity: Performs O(std::distance(first1, last1) + std::distance(first2, last2)) applications of the comparison `comp` and the each projection.

**Template Parameters**

• **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.

• **RandIter1**: The type of the source iterators used (deduced) representing the first sorted range. This iterator type must meet the requirements of an random access iterator.

• **RandIter2**: The type of the source iterators used (deduced) representing the second sorted range. This iterator type must meet the requirements of an random access iterator.

• **RandIter3**: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an random access iterator.

• **Comp**: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of `merge` requires `Comp` to meet the requirements of `CopyConstructible`. This defaults to `std::less<>`

• **Proj1**: The type of an optional projection function to be used for elements of the first range. This defaults to `util::projection_identity`

• **Proj2**: The type of an optional projection function to be used for elements of the second range. This defaults to `util::projection_identity`

2.9. API reference
Parameters

- **policy**: The execution policy to use for the scheduling of the iterations.
- **first1**: Refers to the beginning of the first range of elements the algorithm will be applied to.
- **last1**: Refers to the end of the first range of elements the algorithm will be applied to.
- **first2**: Refers to the beginning of the second range of elements the algorithm will be applied to.
- **last2**: Refers to the end of the second range of elements the algorithm will be applied to.
- **dest**: Refers to the beginning of the destination range.
- **comp**: `comp` is a callable object which returns true if the first argument is less than the second, and false otherwise. The signature of this comparison should be equivalent to:

```cpp
bool comp(const Type1 &a, const Type2 &b);
```

The signature does not need to have `const&`, but the function must not modify the objects passed to it. The types `Type1` and `Type2` must be such that objects of types `RandIter1` and `RandIter2` can be dereferenced and then implicitly converted to both `Type1` and `Type2`.

- **proj1**: Specifies the function (or function object) which will be invoked for each of the elements of the first range as a projection operation before the actual comparison `comp` is invoked.
- **proj2**: Specifies the function (or function object) which will be invoked for each of the elements of the second range as a projection operation before the actual comparison `comp` is invoked.

The assignments in the parallel `merge` algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

**Return** The `merge` algorithm returns a `hpx::future<tagged_tuple<tag::in1(RandIter1),
    tag::in2(RandIter2), tag::out(RandIter3)>>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `tagged_tuple<tag::in1(RandIter1),
    tag::in2(RandIter2), tag::out(RandIter3)>>` otherwise. The `merge` algorithm returns the tuple of the source iterator `last1`, the source iterator `last2`, the destination iterator to the end of the `dest` range.

```
template <typename ExPolicy, typename RandIter, typename Comp = detail::less, typename Proj = util::projection_identity>
util::detail::algorithm_result<ExPolicy, RandIter>::type
inplace_merge(ExPolicy &&policy, RandIter first,
    RandIter middle, RandIter last,
    Comp &&comp = Comp(), Proj &&proj = Proj())
```

Merges two consecutive sorted ranges `[first, middle)` and `[middle, last)` into one sorted range `[first, last)`. The order of equivalent elements in the each of original two ranges is preserved. For equivalent elements in the original two ranges, the elements from the first range precede the elements from the second range.

The assignments in the parallel `inplace_merge` algorithm invoked with an execution policy object of type `sequenced_policy` execute in sequential order in the calling thread.

**Note** Complexity: Performs O(`std::distance(first, last)`) applications of the comparison `comp` and the each projection.

**Template Parameters**

- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- **RandIter**: The type of the source iterators used (deduced). This iterator type must meet
the requirements of an random access iterator.

- **Comp**: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of `inplace_merge` requires `Comp` to meet the requirements of `CopyConstructible`. This defaults to `std::less<>`
- **Proj**: The type of an optional projection function. This defaults to `util::projection_identity`

### Parameters

- **policy**: The execution policy to use for the scheduling of the iterations.
- **first**: Refers to the beginning of the first sorted range the algorithm will be applied to.
- **middle**: Refers to the end of the first sorted range and the beginning of the second sorted range the algorithm will be applied to.
- **last**: Refers to the end of the second sorted range the algorithm will be applied to.
- **comp**: `comp` is a callable object which returns true if the first argument is less than the second, and false otherwise. The signature of this comparison should be equivalent to:

```cpp
bool comp(const Type1 &a, const Type2 &b);
```

The signature does not need to have `const&`, but the function must not modify the objects passed to it. The types `Type1` and `Type2` must be such that objects of types `RandIter` can be dereferenced and then implicitly converted to both `Type1` and `Type2`
- **proj**: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate is invoked.

The assignments in the parallel `inplace_merge` algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

### Return

The `inplace_merge` algorithm returns a `hpx::future<RandIter>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `RandIter` otherwise. The `inplace_merge` algorithm returns the source iterator `last`.

```cpp
#include <hpx/parallel/algorithms/remove.hpp>

namespace hpx

namespace parallel

namespace v1

Functions

```cpp
template <typename ExPolicy, typename FwdIter, typename Pred, typename Proj = util::projection_identity>
util::detail::algorithm_result<ExPolicy, FwdIter>::type
remove_if (ExPolicy &&policy,
    FwdIter first, FwdIter last,
    Pred &&pred, Proj &&proj = Proj())
```

Removes all elements satisfying specific criteria from the range `[first, last)` and returns a past-the-end iterator for the new end of the range. This version removes all elements for which predicate `pred` returns true.

The assignments in the parallel `remove_if` algorithm invoked with an execution policy object of type `sequenced_policy` execute in sequential order in the calling thread.

**Note** Complexity: Performs not more than `last - first` assignments, exactly `last - first` applications of the predicate `pred` and the projection `proj`.

### Template Parameters
• **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.

• **FwdIter**: The type of the source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.

• **Pred**: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of `remove_if` requires `Pred` to meet the requirements of `CopyConstructible`.

• **Proj**: The type of an optional projection function. This defaults to `util::projection_identity`.

## Parameters

- **policy**: The execution policy to use for the scheduling of the iterations.
- **first**: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- **last**: Refers to the end of the sequence of elements the algorithm will be applied to.
- **pred**: Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by `[first, last)`. This is an unary predicate which returns `true` for the required elements. The signature of this predicate should be equivalent to:

  ```
  bool pred(const Type &a);
  ```

  The signature does not need to have `const&`, but the function must not modify the objects passed to it. The type `Type` must be such that an object of type `FwdIter` can be dereferenced and then implicitly converted to `Type`.

- **proj**: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate is invoked.

The assignments in the parallel `remove_if` algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

## Return

The `remove_if` algorithm returns a `hpx::future<FwdIter>` if the execution policy is of type `sequenced_policy` or `parallel_task_policy` and returns `FwdIter` otherwise. The `remove_if` algorithm returns the iterator to the new end of the range.

```
template<typename ExPolicy, typename FwdIter, typename T, typename Proj = util::projection_identity>
util::detail::algorithm_result<ExPolicy, FwdIter>::type
remove(ExPolicy && policy, FwdIter first, FwdIter last, T const &value, Proj &proj = Proj())
```

Removes all elements satisfying specific criteria from the range `[first, last)` and returns a past-the-end iterator for the new end of the range. This version removes all elements that are equal to `value`.

The assignments in the parallel `remove` algorithm invoked with an execution policy object of type `sequenced_policy` execute in sequential order in the calling thread.

**Note** Complexity: Performs not more than `last - first` assignments, exactly `last - first` applications of the operator `==()` and the projection `proj`.

## Template Parameters

- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.

- **FwdIter**: The type of the source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.

- **T**: The type of the value to remove (deduced). This value type must meet the requirements of `CopyConstructible`.

- **Proj**: The type of an optional projection function. This defaults to `util::projection_identity`.

## Parameters

- **policy**: The execution policy to use for the scheduling of the iterations.
• **first**: Refers to the beginning of the sequence of elements the algorithm will be applied to.
• **last**: Refers to the end of the sequence of elements the algorithm will be applied to.
• **value**: Specifies the value of elements to remove.
• **proj**: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate is invoked.

The assignments in the parallel remove algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

**Return** The remove algorithm returns a `hpx::future<FwdIter>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `FwdIter` otherwise. The remove algorithm returns the iterator to the new end of the range.

```cpp
#include <hpx/parallel/algorithms/set_union.hpp>

namespace hpx
{
    namespace parallel
    {
        namespace v1
        {
            Functions
            
            template <typename ExPolicy, typename FwdIter1, typename FwdIter2, typename FwdIter3, typename Pred = detail::less>
            std::enable_if<execution::is_execution_policy<ExPolicy>::value, typename util::detail::algorithm_result<ExPolicy, FwdIter3>::type>::type
            set_union(
                ExPolicy && policy,
                FwdIter1 first1,
                FwdIter1 last1,
                FwdIter2 first2,
                FwdIter2 last2,
                FwdIter3 dest,
                Pred && op = Pred())
            
            Constructs a sorted range beginning at dest consisting of all elements present in one or both sorted ranges \([\text{first}_1, \text{last}_1)\) and \([\text{first}_2, \text{last}_2)\). This algorithm expects both input ranges to be sorted with the given binary predicate \(f\).

            If some element is found \(m\) times in \([\text{first}_1, \text{last}_1)\) and \(n\) times in \([\text{first}_2, \text{last}_2)\), then all \(m\) elements will be copied from \([\text{first}_1, \text{last}_1)\) to dest, preserving order, and then exactly \(\text{std::max}(n-m, 0)\) elements will be copied from \([\text{first}_2, \text{last}_2)\) to dest, also preserving order.

            **Note** Complexity: At most \(2*(N_1 + N_2 - 1)\) comparisons, where \(N_1\) is the length of the first sequence and \(N_2\) is the length of the second sequence.

            The resulting range cannot overlap with either of the input ranges.

            The application of function objects in parallel algorithm invoked with a sequential execution
```
policy object execute in sequential order in the calling thread (sequenced_policy) or in a single new thread spawned from the current thread (for sequenced_task_policy).

**Template Parameters**
- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it applies user-provided function objects.
- **FwdIter1**: The type of the source iterators used (deduced) representing the first sequence. This iterator type must meet the requirements of an forward iterator.
- **FwdIter2**: The type of the source iterators used (deduced) representing the first sequence. This iterator type must meet the requirements of an forward iterator.
- **FwdIter3**: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an output iterator.
- **Op**: The type of an optional function/function object to use. Unlike its sequential form, the parallel overload of set_union requires Pred to meet the requirements of CopyConstructible. This defaults to std::less<>.

**Parameters**
- **policy**: The execution policy to use for the scheduling of the iterations.
- **first1**: Refers to the beginning of the sequence of elements of the first range the algorithm will be applied to.
- **last1**: Refers to the end of the sequence of elements of the first range the algorithm will be applied to.
- **first2**: Refers to the beginning of the sequence of elements of the second range the algorithm will be applied to.
- **last2**: Refers to the end of the sequence of elements of the second range the algorithm will be applied to.
- **dest**: Refers to the beginning of the destination range.
- **op**: The binary predicate which returns true if the elements should be treated as equal. The signature of the predicate function should be equivalent to the following:

```cpp
bool pred(const Type1 &a, const Type1 &b);
```

The signature does not need to have const &, but the function must not modify the objects passed to it. The type Type1 must be such that objects of type InIter can be dereferenced and then implicitly converted to Type1.

The application of function objects in parallel algorithm invoked with an execution policy object of type parallel_policy or parallel_task_policy are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

**Return** The set_union algorithm returns a hpx::future<FwdIter3> if the execution policy is of type sequenced_task_policy or parallel_task_policy and returns FwdIter3 otherwise. The set_union algorithm returns the output iterator to the element in the destination range, one past the last element copied.

```cpp
#include <hpx/parallel/algorithms/minmax.hpp>

namespace hpx {

namespace parallel {

namespace v1 {

template <typename ExPolicy, typename FwdIter1, typename FwdIter2, typename Proj = util::projection_identity, typename F = detail::less>

```
util::detail::algorithm_result<ExPolicy, FwdIter>::type min_element (ExPolicy &&policy, FwdIter first, FwdIter last, F &&f = F(), Proj &&proj = Proj())

Finds the smallest element in the range [first, last) using the given comparison function f.

The comparisons in the parallel min_element algorithm invoked with an execution policy object of type sequenced_policy execute in sequential order in the calling thread.

**Note** Complexity: Exactly \( \max(N-1, 0) \) comparisons, where \( N = \text{std::distance(first, last)} \).

**Template Parameters**
- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- **FwdIter**: The type of the source iterators used (deduced). This iterator type must meet the requirements of a forward iterator.
- **F**: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of min_element requires F to meet the requirements of CopyConstructible.
- **Proj**: The type of an optional projection function. This defaults to util::projection_identity

**Parameters**
- **policy**: The execution policy to use for the scheduling of the iterations.
- **first**: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- **last**: Refers to the end of the sequence of elements the algorithm will be applied to.
- **f**: The binary predicate which returns true if the the left argument is less than the right element. The signature of the predicate function should be equivalent to the following:

```cpp
bool pred(const Type1 &a, const Type1 &b);
```

The signature does not need to have const &, but the function must not modify the objects passed to it. The type Type1 must be such that objects of type FwdIter can be dereferenced and then implicitly converted to Type1.

- **proj**: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate is invoked.

The comparisons in the parallel min_element algorithm invoked with an execution policy object of type parallel_policy or parallel_task_policy are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

**Return** The min_element algorithm returns a hpx::future<FwdIter> if the execution policy is of type sequenced_task_policy or parallel_task_policy and returns FwdIter otherwise. The min_element algorithm returns the iterator to the smallest element in the range [first, last). If several elements in the range are equivalent to the smallest element, returns the iterator to the first such element. Returns last if the range is empty.

```cpp
template <typename ExPolicy, typename FwdIter, typename Proj = util::projection_identity, typename F = util::detail::algorithm_result<ExPolicy, FwdIter>::type> max_element (ExPolicy &&policy, FwdIter first, FwdIter last, F &&f = F(), Proj &&proj = Proj())
```

Finds the greatest element in the range [first, last) using the given comparison function f.

The comparisons in the parallel max_element algorithm invoked with an execution policy object of type sequenced_policy execute in sequential order in the calling thread.

**Note** Complexity: Exactly \( \max(N-1, 0) \) comparisons, where \( N = \text{std::distance(first, last)} \).

**Template Parameters**
- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
• FwdIter: The type of the source iterators used (deduced). This iterator type must meet the requirements of a forward iterator.
• F: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of max_element requires F to meet the requirements of CopyConstructible.
• Proj: The type of an optional projection function. This defaults to util::projection_identity

Parameters
• policy: The execution policy to use for the scheduling of the iterations.
• first: Refers to the beginning of the sequence of elements the algorithm will be applied to.
• last: Refers to the end of the sequence of elements the algorithm will be applied to.
• f: The binary predicate which returns true if the left argument is less than the right element. The signature of the predicate function should be equivalent to the following:

```cpp
bool pred(const Type1 &a, const Type1 &b);
```

The signature does not need to have const &, but the function must not modify the objects passed to it. The type Type1 must be such that objects of type FwdIter can be dereferenced and then implicitly converted to Type1.
• proj: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate is invoked.

The comparisons in the parallel max_element algorithm invoked with an execution policy object of type parallel_policy or parallel_task_policy are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The max_element algorithm returns a hpx::future<FwdIter> if the execution policy is of type sequenced_task_policy or parallel_task_policy and returns FwdIter otherwise. The max_element algorithm returns the iterator to the smallest element in the range [first, last). If several elements in the range are equivalent to the smallest element, returns the iterator to the first such element. Returns last if the range is empty.

```
template <typename ExPolicy, typename FwdIter, typename Proj = util::projection_identity, typename F = hpx::algorithm_result<ExPolicy, tag::min<FwdIter>>, typename FwdIter>>:

FwdIter minmax_element(ExPolicy &policy, FwdIter first, FwdIter last, F &&f
```  

The comparisons in the parallel minmax_element algorithm invoked with an execution policy object of type sequenced_policy execute in sequential order in the calling thread.

Note Complexity: At most max(floor(3/2*(N-1)), 0) applications of the predicate, where N = std::distance(first, last).

Template Parameters
• ExPolicy: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
• FwdIter: The type of the source iterators used (deduced). This iterator type must meet the requirements of a forward iterator.
• F: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of minmax_element requires F to meet the requirements of CopyConstructible.
• Proj: The type of an optional projection function. This defaults to util::projection_identity

Parameters
• policy: The execution policy to use for the scheduling of the iterations.
• first: Refers to the beginning of the sequence of elements the algorithm will be applied to.
• last: Refers to the end of the sequence of elements the algorithm will be applied to.
• `f`: The binary predicate which returns true if the the left argument is less than the right element. This argument is optional and defaults to `std::less`. The signature of the predicate function should be equivalent to the following:

```cpp
bool pred(const Type1 &a, const Type1 &b);
```

The signature does not need to have `const &`, but the function must not modify the objects passed to it. The type `Type1` must be such that objects of type `FwdIter` can be dereferenced and then implicitly converted to `Type1`.

• `proj`: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate is invoked.

The comparisons in the parallel `minmax_element` algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

**Return** The `minmax_element` algorithm returns a `hpx::future<tagged_pair<tag::min(FwdIter), tag::max(FwdIter)>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `tagged_pair<tag::min(FwdIter), tag::max(FwdIter)>` otherwise.

The `minmax_element` algorithm returns a pair consisting of an iterator to the smallest element as the first element and an iterator to the greatest element as the second. Returns `std::make_pair(first, first)` if the range is empty. If several elements are equivalent to the smallest element, the iterator to the first such element is returned. If several elements are equivalent to the largest element, the iterator to the last such element is returned.

```cpp
#include <hpx/parallel/algorithms/set_difference.hpp>

namespace hpx

namespace parallel

namespace v1

Functions

```cpp
template <typename ExPolicy, typename FwdIter1, typename FwdIter2, typename FwdIter3, typename Pred = detail::less>
    std::enable_if<execution::is_execution_policy<ExPolicy>::value, typename util::detail::algorithm_result<ExPolicy, FwdIter3>::type>::type
    set_difference(ExPolicy &&policy, FwdIter1 first1, FwdIter1 last1, FwdIter2 first2, FwdIter2 last2, FwdIter3 dest, Pred &&op = Pred())
```

Constructs a sorted range beginning at `dest` consisting of all elements present in the range `[first1,
last1) and not present in the range [first2, last2). This algorithm expects both input ranges to be sorted with the given binary predicate $f$.

Equivalent elements are treated individually, that is, if some element is found $m$ times in [first1, last1) and $n$ times in [first2, last2), it will be copied to dest exactly $\max(m-n, 0)$ times. The resulting range cannot overlap with either of the input ranges.

**Note** Complexity: At most $2^*(N1 + N2 - 1)$ comparisons, where $N1$ is the length of the first sequence and $N2$ is the length of the second sequence. The resulting range cannot overlap with either of the input ranges.

The application of function objects in parallel algorithm invoked with a sequential execution policy object execute in sequential order in the calling thread (sequence_policy) or in a single new thread spawned from the current thread (for sequenced_task_policy).

**Template Parameters**
- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it applies user-provided function objects.
- **FwdIter1**: The type of the source iterators used (deduced) representing the first sequence. This iterator type must meet the requirements of an forward iterator.
- **FwdIter2**: The type of the source iterators used (deduced) representing the first sequence. This iterator type must meet the requirements of an forward iterator.
- **FwdIter3**: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an output iterator.
- **Pred**: The type of an optional function/function object to use. Unlike its sequential form, the parallel overload of set_difference requires Pred to meet the requirements of CopyConstructible. This defaults to std::less<>

**Parameters**
- **policy**: The execution policy to use for the scheduling of the iterations.
- **first1**: Refers to the beginning of the sequence of elements of the first range the algorithm will be applied to.
- **last1**: Refers to the end of the sequence of elements of the first range the algorithm will be applied to.
- **first2**: Refers to the beginning of the sequence of elements of the second range the algorithm will be applied to.
- **last2**: Refers to the end of the sequence of elements of the second range the algorithm will be applied to.
- **dest**: Refers to the beginning of the destination range.
- **op**: The binary predicate which returns true if the elements should be treated as equal. The signature of the predicate function should be equivalent to the following:

```cpp
bool pred(const Type1 &a, const Type1 &b);
```

The signature does not need to have const &, but the function must not modify the objects passed to it. The type Type1 must be such that objects of type InIter can be dereferenced and then implicitly converted to Type1.

The application of function objects in parallel algorithm invoked with an execution policy object of type parallel_policy or parallel_task_policy are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

**Return** The set_difference algorithm returns a hpx::future<FwdIter3> if the execution policy is of type sequenced_task_policy or parallel_task_policy and returns FwdIter3 otherwise. The set_difference algorithm returns the output iterator to the element in the destination range, one past the last element copied.
```cpp
#include <hpx/parallel/algorithms/for_loop.hpp>

namespace hpx

    namespace parallel

        namespace v2

        Functions

        template <typename ExPolicy, typename I, typename... Args>
        util::detail::algorithm_result<ExPolicy>::type
        for_loop(ExPolicy &&policy, typename std::decay<I>::type first, I last,
                 Args&&... args)

The for_loop implements loop functionality over a range specified by integral or iterator bounds. For the iterator case, these algorithms resemble for_each from the Parallelism TS, but leave to the programmer when and if to dereference the iterator.

Requires: I shall be an integral type or meet the requirements of an input iterator type. The args parameter pack shall have at least one element, comprising objects returned by invocations of reduction and/or induction function templates followed by exactly one element invocable element-access function, f. f shall meet the requirements of MoveConstructible.

Template Parameters

- ExPolicy: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it applies user-provided function objects.
- I: The type of the iteration variable. This could be an (forward) iterator type or an integral type.
- Args: A parameter pack, it’s last element is a function object to be invoked for each iteration, the others have to be either conforming to the induction or reduction concept.

Parameters

- policy: The execution policy to use for the scheduling of the iterations.
- first: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- last: Refers to the end of the sequence of elements the algorithm will be applied to.
- args: The last element of this parameter pack is the function (object) to invoke, while the remaining elements of the parameter pack are instances of either induction or reduction objects. The function (or function object) which will be invoked for each of the elements in the sequence specified by [first, last) should expose a signature equivalent to:

```cpp
<ignored> pred(I const & a, ...);
```

The signature does not need to have const&. It will receive the current value of the iteration variable and one argument for each of the induction or reduction objects passed to the algorithms, representing their current values.

Effects: Applies f to each element in the input sequence, with additional arguments corresponding to the reductions and inductions in the args parameter pack. The length of the input sequence is last - first.

The first element in the input sequence is specified by first. Each subsequent element is generated by incrementing the previous element.

Along with an element from the input sequence, for each member of the args parameter pack excluding f, an additional argument is passed to each application of f as follows:
```
**Note** As described in the C++ standard, arithmetic on non-random-access iterators is performed using *advance* and *distance*.

**Note** The order of the elements of the input sequence is important for determining ordinal position of an application of \( f \), even though the applications themselves may be unordered.

If the pack member is an object returned by a call to a reduction function listed in section, then the additional argument is a reference to a view of that reduction object. If the pack member is an object returned by a call to induction, then the additional argument is the induction value for that induction object corresponding to the position of the application of \( f \) in the input sequence.

Complexity: Applies \( f \) exactly once for each element of the input sequence.

Remarks: If \( f \) returns a result, the result is ignored.

**Return** The *for_loop* algorithm returns a `hpx::future<void>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `void` otherwise.

**template** `<typename I, typename... Args>`

```cpp
void for_loop(typename std::decay<I>::type first, I last, Args&&... args)
```

The *for_loop* implements loop functionality over a range specified by integral or iterator bounds. For the iterator case, these algorithms resemble *for_each* from the Parallelism TS, but leave to the programmer when and if to dereference the iterator.

The execution of *for_loop* without specifying an execution policy is equivalent to specifying `parallel::execution::seq` as the execution policy.

**Requires:** \( I \) shall be an integral type or meet the requirements of an input iterator type. The \( \text{args} \) parameter pack shall have at least one element, comprising objects returned by invocations of reduction and/or induction function templates followed by exactly one element invocable element-access function, \( f \). \( f \) shall meet the requirements of MoveConstructible.

**Template Parameters**

- \( I \): The type of the iteration variable. This could be an (forward) iterator type or an integral type.
- \( \text{Args} \): A parameter pack, it’s last element is a function object to be invoked for each iteration, the others have to be either conforming to the induction or reduction concept.

**Parameters**

- \( \text{first} \): Refers to the beginning of the sequence of elements the algorithm will be applied to.
- \( \text{last} \): Refers to the end of the sequence of elements the algorithm will be applied to.
- \( \text{args} \): The last element of this parameter pack is the function (object) to invoke, while the remaining elements of the parameter pack are instances of either induction or reduction objects. The function (or function object) which will be invoked for each of the elements in the sequence specified by \([\text{first}, \text{last})\) should expose a signature equivalent to:

```cpp
<\text{ignored}> \text{pred}(\text{I const & a}, ...);
```

The signature does not need to have \text{const}&. It will receive the current value of the iteration variable and one argument for each of the induction or reduction objects passed to the algorithms, representing their current values.

**Effects:** Applies \( f \) to each element in the input sequence, with additional arguments corresponding to the reductions and inductions in the \( \text{args} \) parameter pack. The length of the input sequence is \( \text{last} - \text{first} \).

The first element in the input sequence is specified by \( \text{first} \). Each subsequent element is generated by incrementing the previous element.

Along with an element from the input sequence, for each member of the \( \text{args} \) parameter pack excluding \( f \), an additional argument is passed to each application of \( f \) as follows:
Note As described in the C++ standard, arithmetic on non-random-access iterators is performed using advance and distance.

Note The order of the elements of the input sequence is important for determining ordinal position of an application of \( f \), even though the applications themselves may be unordered.

If the pack member is an object returned by a call to a reduction function listed in section, then the additional argument is a reference to a view of that reduction object. If the pack member is an object returned by a call to induction, then the additional argument is the induction value for that induction object corresponding to the position of the application of \( f \) in the input sequence.

Complexity: Applies \( f \) exactly once for each element of the input sequence.

Remarks: If \( f \) returns a result, the result is ignored.

template <typename ExPolicy, typename I, typename S, typename... Args>&&std::is_integral< S >::value>
util::detail::algorithm_result<ExPolicy>::type for_loop_strided (ExPolicy &&policy,
typename
typename
std::decay<I>::type
first, I last, S stride,
Args&&... args)

The for_loop_strided implements loop functionality over a range specified by integral or iterator bounds. For the iterator case, these algorithms resemble for_each from the Parallelism TS, but leave to the programmer when and if to dereference the iterator.

Requires: \( I \) shall be an integral type or meet the requirements of an input iterator type. The \( args \) parameter pack shall have at least one element, comprising objects returned by invocations of reduction and/or induction function templates followed by exactly one element invocable element-access function, \( f \). \( f \) shall meet the requirements of MoveConstructible.

Template Parameters
- \( ExPolicy \): The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it applies user-provided function objects.
- \( I \): The type of the iteration variable. This could be an (forward) iterator type or an integral type.
- \( S \): The type of the stride variable. This should be an integral type.
- \( Args \): A parameter pack, it’s last element is a function object to be invoked for each iteration, the others have to be either conforming to the induction or reduction concept.

Parameters
- \( policy \): The execution policy to use for the scheduling of the iterations.
- \( first \): Refers to the beginning of the sequence of elements the algorithm will be applied to.
- \( last \): Refers to the end of the sequence of elements the algorithm will be applied to.
- \( stride \): Refers to the stride of the iteration steps. This shall have non-zero value and shall be negative only if \( I \) has integral type or meets the requirements of a bidirectional iterator.
- \( args \): The last element of this parameter pack is the function (object) to invoke, while the remaining elements of the parameter pack are instances of either induction or reduction objects. The function (or function object) which will be invoked for each of the elements in the sequence specified by \([first, last)\) should expose a signature equivalent to:

\[
\text{<ignored> pred(I const & a, ...);}\]

The signature does not need to have const&. It will receive the current value of the iteration variable and one argument for each of the induction or reduction objects passed to the algorithms, representing their current values.

Effects: Applies \( f \) to each element in the input sequence, with additional arguments corresponding to the reductions and inductions in the \( args \) parameter pack. The length of the input sequence is \( last - first \).
The first element in the input sequence is specified by `first`. Each subsequent element is generated by incrementing the previous element.

Along with an element from the input sequence, for each member of the `args` parameter pack excluding `f`, an additional argument is passed to each application of `f` as follows:

**Note** As described in the C++ standard, arithmetic on non-random-access iterators is performed using advance and distance.

**Note** The order of the elements of the input sequence is important for determining ordinal position of an application of `f`, even though the applications themselves may be unordered.

If the pack member is an object returned by a call to a reduction function listed in section, then the additional argument is a reference to a view of that reduction object. If the pack member is an object returned by a call to induction, then the additional argument is the induction value for that induction object corresponding to the position of the application of `f` in the input sequence.

Complexity: Applies `f` exactly once for each element of the input sequence.

Remarks: If `f` returns a result, the result is ignored.

**Return** The `for_loop_strided` algorithm returns a `hpx::future<void>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `void` otherwise.

```cpp
template <typename I, typename S, typename... Args, &&std::is_integral< S >::value>
void for_loop_strided(I std::decay<I>::type first, I last, S stride, Args&&... args)
```

The `for_loop_strided` implements loop functionality over a range specified by integral or iterator bounds. For the iterator case, these algorithms resemble for_each from the Parallelism TS, but leave to the programmer when and if to dereference the iterator.

The execution of `for_loop` without specifying an execution policy is equivalent to specifying `parallel::execution::seq` as the execution policy.

Requires: `I` shall be an integral type or meet the requirements of an input iterator type. The `args` parameter pack shall have at least one element, comprising objects returned by invocations of `reduction` and/or `induction` function templates followed by exactly one element invocable element-access function, `f`. `f` shall meet the requirements of MoveConstructible.

**Template Parameters**
- `I`: The type of the iteration variable. This could be an (forward) iterator type or an integral type.
- `S`: The type of the stride variable. This should be an integral type.
- `Args`: A parameter pack, it’s last element is a function object to be invoked for each iteration, the others have to be either conforming to the induction or reduction concept.

**Parameters**
- `first`: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- `last`: Refers to the end of the sequence of elements the algorithm will be applied to.
- `stride`: Refers to the stride of the iteration steps. This shall have non-zero value and shall be negative only if `I` has integral type or meets the requirements of a bidirectional iterator.
- `args`: The last element of this parameter pack is the function (object) to invoke, while the remaining elements of the parameter pack are instances of either induction or reduction objects. The function (or function object) which will be invoked for each of the elements in the sequence specified by `[first, last)` should expose a signature equivalent to:

```cpp
<ignored> pred(I const& a, ...);
```

The signature does not need to have `const&`. It will receive the current value of the iteration variable and one argument for each of the induction or reduction objects passed to the algorithms, representing their current values.
Effects: Applies \( f \) to each element in the input sequence, with additional arguments corresponding to the reductions and inductions in the \( \textit{args} \) parameter pack. The length of the input sequence is \( \text{last} - \text{first} \).

The first element in the input sequence is specified by \( \text{first} \). Each subsequent element is generated by incrementing the previous element.

Along with an element from the input sequence, for each member of the \( \textit{args} \) parameter pack excluding \( f \), an additional argument is passed to each application of \( f \) as follows:

\textbf{Note} As described in the C++ standard, arithmetic on non-random-access iterators is performed using advance and distance.

\textbf{Note} The order of the elements of the input sequence is important for determining ordinal position of an application of \( f \), even though the applications themselves may be unordered.

If the pack member is an object returned by a call to a reduction function listed in section, then the additional argument is a reference to a view of that reduction object. If the pack member is an object returned by a call to induction, then the additional argument is the induction value for that induction object corresponding to the position of the application of \( f \) in the input sequence.

Complexity: Applies \( f \) exactly once for each element of the input sequence.

Remarks: If \( f \) returns a result, the result is ignored.

\texttt{template <typename ExPolicy, typename I, typename Size, typename... Args> &&std::is_integral< Size >::value>}
\texttt{util::detail::algorithm_result<ExPolicy>::type for_loop_n (ExPolicy &&policy, I \textit{first}, Size size, Args&&... \textit{args})}

The \texttt{for_loop_n} implements loop functionality over a range specified by integral or iterator bounds. For the iterator case, these algorithms resemble \texttt{for_each} from the Parallelism TS, but leave to the programmer when and if to dereference the iterator.

Requires: \( I \) shall be an integral type or meet the requirements of an input iterator type. The \( \textit{args} \) parameter pack shall have at least one element, comprising objects returned by invocations of \textit{reduction} and/or \textit{induction} function templates followed by exactly one element invokeable element-access function, \( f \). \( f \) shall meet the requirements of MoveConstructible.

\textbf{Template Parameters}
- \( \textit{ExPolicy} \): The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it applies user-provided function objects.
- \( I \): The type of the iteration variable. This could be an (forward) iterator type or an integral type.
- \( Size \): The type of a non-negative integral value specifying the number of items to iterate over.
- \( \textit{Args} \): A parameter pack, it’s last element is a function object to be invoked for each iteration, the others have to be either conforming to the induction or reduction concept.

\textbf{Parameters}
- \( \textit{policy} \): The execution policy to use for the scheduling of the iterations.
- \( \textit{first} \): Refers to the beginning of the sequence of elements the algorithm will be applied to.
- \( \textit{size} \): Refers to the number of items the algorithm will be applied to.
- \( \textit{args} \): The last element of this parameter pack is the function (object) to invoke, while the remaining elements of the parameter pack are instances of either induction or reduction objects. The function (or function object) which will be invoked for each of the elements in the sequence specified by \( \text{first}, \text{last} \) should expose a signature equivalent to:

\[
\text{<ignored>} \text{ pred}(I \text{ const & a, ...});
\]

The signature does not need to have \text{const &}. It will receive the current value of the iteration variable and one argument for each of the induction or reduction objects passed to the
algorithms, representing their current values.

Effects: Applies \( f \) to each element in the input sequence, with additional arguments corresponding to the reductions and inductions in the \( args \) parameter pack. The length of the input sequence is \( \text{last - first} \).

The first element in the input sequence is specified by \( \text{first} \). Each subsequent element is generated by incrementing the previous element.

Along with an element from the input sequence, for each member of the \( args \) parameter pack excluding \( f \), an additional argument is passed to each application of \( f \) as follows:

**Note** As described in the C++ standard, arithmetic on non-random-access iterators is performed using advance and distance.

**Note** The order of the elements of the input sequence is important for determining ordinal position of an application of \( f \), even though the applications themselves may be unordered.

If the pack member is an object returned by a call to a reduction function listed in section, then the additional argument is a reference to a view of that reduction object. If the pack member is an object returned by a call to induction, then the additional argument is the induction value for that induction object corresponding to the position of the application of \( f \) in the input sequence.

Complexity: Applies \( f \) exactly once for each element of the input sequence.

Remarks: If \( f \) returns a result, the result is ignored.

**Return** The \( \text{for_loop_n} \) algorithm returns a \( \text{hpx::future<void>} \) if the execution policy is of type \( \text{sequenced\_task\_policy or parallel\_task\_policy} \) and returns \( \text{void} \) otherwise.

```cpp
template <typename I, typename Size, typename... Args>
void for_loop_n(I first, Size size, Args&&... args)
```

The \( \text{for_loop} \) implements loop functionality over a range specified by integral or iterator bounds. For the iterator case, these algorithms resemble \( \text{for\_each} \) from the Parallelism TS, but leave to the programmer when and if to dereference the iterator.

The execution of \( \text{for\_loop} \) without specifying an execution policy is equivalent to specifying \( \text{parallel::execution::seq} \) as the execution policy.

Requires: \( I \) shall be an integral type or meet the requirements of an input iterator type. The \( args \) parameter pack shall have at least one element, comprising objects returned by invocations of reduction and/or induction function templates followed by exactly one element invocable element-access function, \( f \). \( f \) shall meet the requirements of MoveConstructible.

**Template Parameters**
- \( I \): The type of the iteration variable. This could be an (forward) iterator type or an integral type.
- \( Size \): The type of a non-negative integral value specifying the number of items to iterate over.
- \( Args \): A parameter pack, it’s last element is a function object to be invoked for each iteration, the others have to be either conforming to the induction or reduction concept.

**Parameters**
- \( \text{first} \): Refers to the beginning of the sequence of elements the algorithm will be applied to.
- \( \text{size} \): Refers to the number of items the algorithm will be applied to.
- \( \text{args} \): The last element of this parameter pack is the function (object) to invoke, while the remaining elements of the parameter pack are instances of either induction or reduction objects. The function (or function object) which will be invoked for each of the elements in the sequence specified by \( [\text{first}, \text{last}) \) should expose a signature equivalent to:

```cpp
<\text{ignored}> \text{pred}(I \text{ const\& a, ...});
```

The signature does not need to have \text{const\&}. It will receive the current value of the itera-
tion variable and one argument for each of the induction or reduction objects passed to the algorithms, representing their current values.

Effects: Applies $f$ to each element in the input sequence, with additional arguments corresponding to the reductions and inductions in the $args$ parameter pack. The length of the input sequence is $last - first$.

The first element in the input sequence is specified by $first$. Each subsequent element is generated by incrementing the previous element.

Along with an element from the input sequence, for each member of the $args$ parameter pack excluding $f$, an additional argument is passed to each application of $f$ as follows:

**Note** As described in the C++ standard, arithmetic on non-random-access iterators is performed using advance and distance.

**Note** The order of the elements of the input sequence is important for determining ordinal position of an application of $f$, even though the applications themselves may be unordered.

If the pack member is an object returned by a call to a reduction function listed in section, then the additional argument is a reference to a view of that reduction object. If the pack member is an object returned by a call to induction, then the additional argument is the induction value for that induction object corresponding to the position of the application of $f$ in the input sequence.

Complexity: Applies $f$ exactly once for each element of the input sequence.

Remarks: If $f$ returns a result, the result is ignored.

```cpp
template<typename ExPolicy, typename I, typename Size, typename S, typename... Args, &&std::is_integral< Size >::value &&std::is_integral< S >::value>
util::detail::algorithm_result<ExPolicy>::type
for_loop_n_strided(ExPolicy &&policy,
    I first, Size size, S stride,
    Args&&... args)
```

The for_loop_n_strided implements loop functionality over a range specified by integral or iterator bounds. For the iterator case, these algorithms resemble for_each from the Parallelism TS, but leave to the programmer when and if to dereference the iterator.

Requires: $I$ shall be an integral type or meet the requirements of an input iterator type. The $args$ parameter pack shall have at least one element, comprising objects returned by invocations of reduction and/or induction function templates followed by exactly one element invocable element-access function, $f$. $f$ shall meet the requirements of MoveConstructible.

**Template Parameters**

- $ExPolicy$: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it applies user-provided function objects.
- $I$: The type of the iteration variable. This could be an (forward) iterator type or an integral type.
- $Size$: The type of a non-negative integral value specifying the number of items to iterate over.
- $S$: The type of the stride variable. This should be an integral type.
- $Args$: A parameter pack, it’s last element is a function object to be invoked for each iteration, the others have to be either conforming to the induction or reduction concept.

**Parameters**

- $policy$: The execution policy to use for the scheduling of the iterations.
- $first$: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- $size$: Refers to the number of items the algorithm will be applied to.
- $stride$: Refers to the stride of the iteration steps. This shall have non-zero value and shall be negative only if $I$ has integral type or meets the requirements of a bidirectional iterator.
- $args$: The last element of this parameter pack is the function (object) to invoke, while the remaining elements of the parameter pack are instances of either induction or reduction.
objects. The function (or function object) which will be invoked for each of the elements in
the sequence specified by [first, last) should expose a signature equivalent to:

```cpp
<ignored> pred(I const & a, ...);
```

The signature does not need to have const&. It will receive the current value of the iteration
variable and one argument for each of the induction or reduction objects passed to the
algorithms, representing their current values.

Effects: Applies \( f \) to each element in the input sequence, with additional arguments corresponding
to the reductions and inductions in the \( \text{args} \) parameter pack. The length of the input sequence is
last - first.

The first element in the input sequence is specified by \( \text{first} \). Each subsequent element is generated
by incrementing the previous element.

Along with an element from the input sequence, for each member of the \( \text{args} \) parameter pack
excluding \( f \), an additional argument is passed to each application of \( f \) as follows:

**Note** As described in the C++ standard, arithmetic on non-random-access iterators is performed
using advance and distance.

**Note** The order of the elements of the input sequence is important for determining ordinal posi-
tion of an application of \( f \), even though the applications themselves may be unordered.
If the pack member is an object returned by a call to a reduction function listed in section, then
the additional argument is a reference to a view of that reduction object. If the pack member is an
object returned by a call to induction, then the additional argument is the induction value for that
induction object corresponding to the position of the application of \( f \) in the input sequence.

Complexity: Applies \( f \) exactly once for each element of the input sequence.

Remarks: If \( f \) returns a result, the result is ignored.

**Return** The \( \text{for_loop_n_strided} \) algorithm returns a \( \text{hpx::future<void>} \) if the execution policy is
of type \( \text{sequenced_task_policy} \) or \( \text{parallel_task_policy} \) and returns \( \text{void} \) otherwise.

```cpp
template<typename I, typename Size, typename S, typename... Args>
void for_loop_n_strided(I first, Size size, S stride, Args&&... args)
```

The \( \text{for_loop_n_strided} \) implements loop functionality over a range specified by integral or iter-
ator bounds. For the iterator case, these algorithms resemble \( \text{for_each} \) from the Parallelism TS,
but leave to the programmer when and if to dereference the iterator.

The execution of \( \text{for_loop} \) without specifying an execution policy is equivalent to specifying
\( \text{parallel::execution::seq} \) as the execution policy.

Requires: \( I \) shall be an integral type or meet the requirements of an input iterator type. The \( \text{args} \)
parameter pack shall have at least one element, comprising objects returned by invocations of \( \text{re-
duction} \) and/or \( \text{induction} \) function templates followed by exactly one element invocable element-
access function, \( f \). \( f \) shall meet the requirements of \text{MoveConstructible}.

**Template Parameters**
- \( I \): The type of the iteration variable. This could be an (forward) iterator type or an integral
type.
- \( \text{Size} \): The type of a non-negative integral value specifying the number of items to iterate
over.
- \( S \): The type of the stride variable. This should be an integral type.
- \( \text{Args} \): A parameter pack, it’s last element is a function object to be invoked for each itera-
tion, the others have to be either conforming to the induction or reduction concept.

**Parameters**
- \( \text{first} \): Refers to the beginning of the sequence of elements the algorithm will be applied
to.
- \( \text{size} \): Refers to the number of items the algorithm will be applied to.
• **stride**: Refers to the stride of the iteration steps. This shall have non-zero value and shall be negative only if I has integral type or meets the requirements of a bidirectional iterator.

• **args**: The last element of this parameter pack is the function (object) to invoke, while the remaining elements of the parameter pack are instances of either induction or reduction objects. The function (or function object) which will be invoked for each of the elements in the sequence specified by [first, last) should expose a signature equivalent to:

```cpp
<ignored> pred(I const & a, ...);
```

The signature does not need to have const&. It will receive the current value of the iteration variable and one argument for each of the induction or reduction objects passed to the algorithms, representing their current values.

**Effects**: Applies f to each element in the input sequence, with additional arguments corresponding to the reductions and inductions in the args parameter pack. The length of the input sequence is last - first.

The first element in the input sequence is specified by first. Each subsequent element is generated by incrementing the previous element.

Along with an element from the input sequence, for each member of the args parameter pack excluding f, an additional argument is passed to each application of f as follows:

**Note** As described in the C++ standard, arithmetic on non-random-access iterators is performed using advance and distance.

**Note** The order of the elements of the input sequence is important for determining ordinal position of an application of f, even though the applications themselves may be unordered.

If the pack member is an object returned by a call to a reduction function listed in section, then the additional argument is a reference to a view of that reduction object. If the pack member is an object returned by a call to induction, then the additional argument is the induction value for that induction object corresponding to the position of the application of f in the input sequence.

**Complexity**: Applies f exactly once for each element of the input sequence.

**Remarks**: If f returns a result, the result is ignored.

```cpp
#include <hpx/parallel/algorithms/transform.hpp>

namespace hpx
{

namespace parallel
{

namespace v1
{

Functions

template <typename ExPolicy, typename FwdIter1, typename FwdIter2, typename F, typename Proj = util::projection_identity>
util::detail::algorithm_result<ExPolicy, hpx::util::tagged_pair<tag::in(FwdIter1), tag::out FwdIter2>>::type
transform(ExPolicy && policy, FwdIter1 first, FwdIter1 last, FwdIter2 dest, F && f, Proj && proj = Proj())
Applies the given function f to the range [first, last) and stores the result in another range, beginning at dest.

The invocations of f in the parallel transform algorithm invoked with an execution policy object of type sequenced_policy execute in sequential order in the calling thread.

**Note** Complexity: Exactly last - first applications of f

---

2.9. API reference 805
• **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the invocations of $f$.

• **FwdIter1**: The type of the source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.

• **FwdIter2**: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an forward iterator.

• **$F$**: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of `transform` requires $F$ to meet the requirements of `CopyConstructible`.

• **Proj**: The type of an optional projection function. This defaults to `util::projection_identity`.

**Parameters**

• **policy**: The execution policy to use for the scheduling of the iterations.

• **first**: Refers to the beginning of the sequence of elements the algorithm will be applied to.

• **last**: Refers to the end of the sequence of elements the algorithm will be applied to.

• **dest**: Refers to the beginning of the destination range.

• **$f$**: Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by `[first, last)`. This is an unary predicate. The signature of this predicate should be equivalent to:

```cpp
Ret fun(const Type &a);
```

The signature does not need to have `const&`. The type `Type` must be such that an object of type `FwdIter` can be dereferenced and then implicitly converted to `Type`. The type `Ret` must be such that an object of type `FwdIter2` can be dereferenced and assigned a value of type `Ret`.

• **proj**: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate $f$ is invoked.

The invocations of $f$ in the parallel `transform` algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

**Return** The `transform` algorithm returns a `hpx::future<tagged_pair<tag::in(FwdIter1), tag::out(FwdIter2)>>` if the execution policy is of type `parallel_task_policy` and returns `tagged_pair<tag::in(FwdIter1), tag::out(FwdIter2)>>` otherwise. The `transform` algorithm returns a tuple holding an iterator referring to the first element after the input sequence and the output iterator to the element in the destination range, one past the last element copied.

**template**<typename ExPolicy, typename FwdIter1, typename FwdIter2, typename FwdIter3, typename Proj1, typename Proj2 = util::projection_identity>

```cpp
util::detail::algorithm_result<ExPolicy, hpx::util::tagged_tuple<tag::in1(FwdIter1), tag::out(FwdIter2)>, tag::in2(FwdIter2), tag::out(FwdIter3)>::type transform
```

Applies the given function $f$ to pairs of elements from two ranges: one defined by `[first1, last1)` and the other beginning at `first2`, and stores the result in another range, beginning at `dest`.

The invocations of $f$ in the parallel `transform` algorithm invoked with an execution policy object of type `sequenced_policy` execute in sequential order in the calling thread.

**Note** Complexity: Exactly `last - first` applications of $f$.

**Template Parameters**

• **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the invocations of $f$.

• **FwdIter1**: The type of the source iterators for the first range used (deduced). This iterator type must meet the requirements of an forward iterator.

• **FwdIter2**: The type of the source iterators for the second range used (deduced). This iterator type must meet the requirements of an forward iterator.
• FwdIter3: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an forward iterator.

• F: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of transform requires F to meet the requirements of CopyConstructible.

• Proj1: The type of an optional projection function to be used for elements of the first sequence. This defaults to util::projection_identity

• Proj2: The type of an optional projection function to be used for elements of the second sequence. This defaults to util::projection_identity

**Parameters**

- **policy**: The execution policy to use for the scheduling of the iterations.
- **first1**: Refers to the beginning of the first sequence of elements the algorithm will be applied to.
- **last1**: Refers to the end of the first sequence of elements the algorithm will be applied to.
- **first2**: Refers to the beginning of the second sequence of elements the algorithm will be applied to.
- **dest**: Refers to the beginning of the destination range.
- **f**: Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by [first, last). This is a binary predicate. The signature of this predicate should be equivalent to:

  ```cpp
  Ret fun(const Type1 &a, const Type2 &b);
  ```

  The signature does not need to have const&. The types Type1 and Type2 must be such that objects of types FwdIter1 and FwdIter2 can be dereferenced and then implicitly converted to Type1 and Type2 respectively. The type Ret must be such that an object of type FwdIter3 can be dereferenced and assigned a value of type Ret.

- **proj1**: Specifies the function (or function object) which will be invoked for each of the elements of the first sequence as a projection operation before the actual predicate f is invoked.

- **proj2**: Specifies the function (or function object) which will be invoked for each of the elements of the second sequence as a projection operation before the actual predicate f is invoked.

The invocations of f in the parallel transform algorithm invoked with an execution policy object of type parallel_policy or parallel_task_policy are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

**Return**

The transform algorithm returns a hpx::future<tagged_tuple<tag::in1(FwdIter1), tag::in2(FwdIter2), tag::out(FwdIter3)> if the execution policy is of type parallel_task_policy and returns tagged_tuple<tag::in1(FwdIter1), tag::in2(FwdIter2), tag::out(FwdIter3)> otherwise. The transform algorithm returns a tuple holding an iterator referring to the first element after the first input sequence, an iterator referring to the first element after the second input sequence, and the output iterator referring to the element in the destination range, one past the last element copied.

```cpp
template <typename ExPolicy, typename FwdIter1, typename FwdIter2, typename FwdIter3, typename F, typename Proj1 = util::projection_identity, typename Proj2 = util::projection_identity> applies the given function f to pairs of elements from two ranges: one defined by [first1, last1) and the other beginning at first2, and stores the result in another range, beginning at dest.

The invocations of f in the parallel transform algorithm invoked with an execution policy object of type sequenced_policy execute in sequential order in the calling thread.

**Note** Complexity: Exactly min(last2-first2, last1-first1) applications of f

**Template Parameters**

- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in
which the execution of the algorithm may be parallelized and the manner in which it executes
the invocations of \( f \).

- **FwdIter1**: The type of the source iterators for the first range used (deduced). This iterator
type must meet the requirements of an forward iterator.
- **FwdIter2**: The type of the source iterators for the second range used (deduced). This
iterator type must meet the requirements of an forward iterator.
- **FwdIter3**: The type of the iterator representing the destination range (deduced). This
iterator type must meet the requirements of an forward iterator.
- **F**: The type of the function/function object to use (deduced). Unlike its sequential form, the
parallel overload of `transform` requires \( F \) to meet the requirements of `CopyConstructible`.
- **Proj1**: The type of an optional projection function to be used for elements of the first
sequence. This defaults to `util::projection_identity`
- **Proj2**: The type of an optional projection function to be used for elements of the second
sequence. This defaults to `util::projection_identity`

**Parameters**

- **policy**: The execution policy to use for the scheduling of the iterations.
- **first1**: Refers to the beginning of the first sequence of elements the algorithm will be
applied to.
- **last1**: Refers to the end of the first sequence of elements the algorithm will be applied to.
- **first2**: Refers to the beginning of the second sequence of elements the algorithm will be
applied to.
- **last2**: Refers to the end of the second sequence of elements the algorithm will be applied to.
- **dest**: Refers to the beginning of the destination range.
- **f**: Specifies the function (or function object) which will be invoked for each of the elements
in the sequence specified by \([\text{first}, \text{last})\). This is a binary predicate. The signature of this
predicate should be equivalent to:

\[
\text{Ret } \text{fun}(\text{const Type1 } &a, \text{ const Type2 } &b);
\]

The signature does not need to have const&. The types `Type1` and `Type2` must be such that
objects of types `FwdIter1` and `FwdIter2` can be dereferenced and then implicitly converted
to `Type1` and `Type2` respectively. The type `Ret` must be such that an object of type `FwdIter3`
can be dereferenced and assigned a value of type `Ret`.
- **proj1**: Specifies the function (or function object) which will be invoked for each of the ele-
ments of the first sequence as a projection operation before the actual predicate \( f \) is invoked.
- **proj2**: Specifies the function (or function object) which will be invoked for each of the
elements of the second sequence as a projection operation before the actual predicate \( f \) is
invoked.

The invocations of \( f \) in the parallel `transform` algorithm invoked with an execution policy object
of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion
in unspecified threads, and indeterminately sequenced within each thread.

**Note** The algorithm will invoke the binary predicate until it reaches the end of the shorter of the
two given input sequences.

**Return** The `transform` algorithm returns a `hpx::future<tagged_tuple<tag::in1(FwdIter1),
tag::in2(FwdIter2), tag::out(FwdIter3)>>` if the execution policy is of type `parallel_policy` and returns `tagged_tuple<tag::in1(FwdIter1), tag::in2(FwdIter2), tag::out(FwdIter3)>>` otherwise. The `transform` algorithm returns a tuple holding an iterator referring to the first element after the first input sequence, an iterator referring to the first
element after the second input sequence, and the output iterator referring to the element in the
destination range, one past the last element copied.
#include <hpx/parallel/algorithms/unique.hpp>

namespace hpx

namespace parallel

namespace v1

Functions

```cpp
template <typename ExPolicy, typename FwdIter, typename Pred = detail::equal_to, typename Proj = util::projection_identity>
util::detail::algorithm_result<ExPolicy, FwdIter>::type
unique(ExPolicy &&policy, FwdIter first, FwdIter last, Pred &&pred = Pred(), Proj &&proj = Proj())
```

Eliminates all but the first element from every consecutive group of equivalent elements from the range [first, last) and returns a past-the-end iterator for the new logical end of the range.

The assignments in the parallel unique algorithm invoked with an execution policy object of type sequenced_policy execute in sequential order in the calling thread.

**Note** Complexity: Performs not more than last - first assignments, exactly last - first - 1 applications of the predicate pred and no more than twice as many applications of the projection proj.

**Template Parameters**
- ExPolicy: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- FwdIter: The type of the source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.
- Pred: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of unique requires Pred to meet the requirements of CopyConstructible. This defaults to std::equal_to<>
- Proj: The type of an optional projection function. This defaults to util::projection_identity

**Parameters**
- policy: The execution policy to use for the scheduling of the iterations.
- first: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- last: Refers to the end of the sequence of elements the algorithm will be applied to.
- pred: Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by [first, last). This is an binary predicate which returns true for the required elements. The signature of this predicate should be equivalent to:

  ```cpp
  bool pred(const Type1 &a, const Type2 &b);
  ```

  The signature does not need to have const&, but the function must not modify the objects passed to it. The types Type1 and Type2 must be such that objects of types FwdIter can be dereferenced and then implicitly converted to both Type1 and Type2
- proj: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate is invoked.

The assignments in the parallel unique algorithm invoked with an execution policy object of type parallel_policy or parallel_task_policy are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

**Return** The unique algorithm returns a hpx::future<FwdIter> if the execution policy is of type
sequence_task_policy or parallel_task_policy and returns FwdIter otherwise. The unique algorithm returns the iterator to the new end of the range.

```cpp
template <typename ExPolicy, typename FwdIter1, typename FwdIter2, typename Pred = detail::equal_to,
          typename Proj = util::projection_identity>
util::detail::algorithm_result<ExPolicy, hpx::util::tagged_pair<tag::in(FwdIter1), tag::out(FwdIter2)>>::type unique_copy
ExPolicy &&policy, FwdIter1 first, FwdIter1 last, FwdIter2 dest, Pred &&pred = Pred(), Proj &&proj = Proj()Copies the elements from the range [first, last), to another range beginning at dest in such a way that there are no consecutive equal elements. Only the first element of each group of equal elements is copied.
```

The assignments in the parallel unique_copy algorithm invoked with an execution policy object of type sequenced_policy execute in sequential order in the calling thread.

**Note** Complexity: Performs not more than last - first assignments, exactly last - first - 1 applications of the predicate pred and no more than twice as many applications of the projection proj

**Template Parameters**
- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- **FwdIter1**: The type of the source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.
- **FwdIter2**: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an forward iterator.
- **Pred**: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of unique_copy requires Pred to meet the requirements of CopyConstructible. This defaults to std::equal_to<>
- **Proj**: The type of an optional projection function. This defaults to util::projection_identity

**Parameters**
- **policy**: The execution policy to use for the scheduling of the iterations.
- **first**: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- **last**: Refers to the end of the sequence of elements the algorithm will be applied to.
- **dest**: Refers to the beginning of the destination range.
- **pred**: Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by [first, last). This is an binary predicate which returns true for the required elements. The signature of this predicate should be equivalent to:

```cpp
bool pred(const Type &a, const Type &b);
```

The signature does not need to have const&, but the function must not modify the objects passed to it. The type Type must be such that an object of type FwdIter1 can be dereferenced and then implicitly converted to Type.
- **proj**: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate is invoked.

The assignments in the parallel unique_copy algorithm invoked with an execution policy object of type parallel_policy or parallel_task_policy are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

**Return** The unique_copy algorithm returns a hpx::future<tagged_pair<tag::in(FwdIter1), tag::out(FwdIter2)>> if the execution policy is of type sequenced_task_policy or parallel_task_policy and returns tagged_pair<tag::in(FwdIter1), tag::out(FwdIter2)> otherwise. The unique_copy algorithm returns the pair of the source iterator to last, and the destination iterator to the end of the dest range.
#include <hpx/parallel/algorithms/for_loop_reduction.hpp>

namespace hpx

namespace parallel

namespace v2

Functions

template<typename T, typename Op>
detail::reduction_helper<T, typename std::decay<Op>::type> reduction(T &var, T const &identity, Op &&combiner)

The function template returns a reduction object of unspecified type having a value type and encapsulating an identity value for the reduction, a combiner function object, and a live-out object from which the initial value is obtained and into which the final value is stored.

A parallel algorithm uses reduction objects by allocating an unspecified number of instances, called views, of the reduction’s value type. Each view is initialized with the reduction object’s identity value, except that the live-out object (which was initialized by the caller) comprises one of the views. The algorithm passes a reference to a view to each application of an element-access function, ensuring that no two concurrently-executing invocations share the same view. A view can be shared between two applications that do not execute concurrently, but initialization is performed only once per view.

Modifications to the view by the application of element access functions accumulate as partial results. At some point before the algorithm returns, the partial results are combined, two at a time, using the reduction object’s combiner operation until a single value remains, which is then assigned back to the live-out object.

T shall meet the requirements of CopyConstructible and MoveAssignable. The expression var = combiner(var, var) shall be well formed.

Template Parameters

- **T**: The value type to be used by the induction object.
- **Op**: The type of the binary function (object) used to perform the reduction operation.

Parameters

- **var**: [in,out] The life-out value to use for the reduction object. This will hold the reduced value after the algorithm is finished executing.
- **identity**: [in] The identity value to use for the reduction operation.
- **combiner**: [in] The binary function (object) used to perform a pairwise reduction on the elements.

Note In order to produce useful results, modifications to the view should be limited to commutative operations closely related to the combiner operation. For example if the combiner is plus<T>, incrementing the view would be consistent with the combiner but doubling it or assigning to it would not.

Return This returns a reduction object of unspecified type having a value type of T. When the return value is used by an algorithm, the reference to var is used as the live-out object, new views are initialized to a copy of identity, and views are combined by invoking the copy of combiner, passing it the two views to be combined.
#include <hpx/parallel/algorithms/replace.hpp>

namespace hpx

namespace parallel

namespace v1

Functions

template <typename ExPolicy, typename FwdIter, typename T1, typename T2, typename Proj = util::projection_identity>
util::detail::algorithm_result<ExPolicy, FwdIter>::type replace(ExPolicy &&policy, FwdIter first, FwdIter last, T1 const &old_value, T2 const &new_value, Proj &&proj = Proj())

Replaces all elements satisfying specific criteria with new_value in the range [first, last).

Effects: Substitutes elements referred by the iterator it in the range [first, last) with new_value, when the following corresponding conditions hold: INVOKE(proj, *it) == old_value

The assignments in the parallel replace algorithm invoked with an execution policy object of type sequenced_policy execute in sequential order in the calling thread.

Note Complexity: Performs exactly last - first assignments.

Template Parameters

• ExPolicy: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
• FwdIter: The type of the source iterators used (deduced). This iterator type must meet the requirements of a forward iterator.
• T1: The type of the old value to replace (deduced).
• T2: The type of the new values to replace (deduced).
• Proj: The type of an optional projection function. This defaults to util::projection_identity

Parameters

• policy: The execution policy to use for the scheduling of the iterations.
• first: Refers to the beginning of the sequence of elements the algorithm will be applied to.
• last: Refers to the end of the sequence of elements the algorithm will be applied to.
• old_value: Refers to the old value of the elements to replace.
• new_value: Refers to the new value to use as the replacement.
• proj: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate is invoked.

The assignments in the parallel replace algorithm invoked with an execution policy object of type parallel_policy or parallel_task_policy are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The replace algorithm returns a hpx::future<FwdIter> if the execution policy is of type sequenced_task_policy or parallel_task_policy and returns void otherwise. It returns last.

template <typename ExPolicy, typename FwdIter, typename F, typename T, typename Proj = util::projection_identity>
util::detail::algorithm_result<ExPolicy, FwdIter>::type replace_if(ExPolicy &&policy, FwdIter first, FwdIter last, F &&f, T const &new_value, Proj &&proj = Proj())
Replaces all elements satisfying specific criteria (for which predicate \( f \) returns true) with \( \text{new\_value} \) in the range \([\text{first}, \text{last})\).

Effects: Substitutes elements referred by the iterator \( it \) in the range \([\text{first}, \text{last})\) with \( \text{new\_value} \), when the following corresponding conditions hold: \( \text{INVOTE}(f, \text{INVOTE}(\text{proj}, *it)) 
eq \text{false} \)

The assignments in the parallel replace_if algorithm invoked with an execution policy object of type sequenced_policy execute in sequential order in the calling thread.

**Note** Complexity: Performs exactly \( \text{last} - \text{first} \) applications of the predicate.

**Template Parameters**
- \( \text{ExPolicy} \): The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- \( \text{FwdIter} \): The type of the source iterators used (deduced). This iterator type must meet the requirements of a forward iterator.
- \( F \): The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of equal requires \( F \) to meet the requirements of CopyConstructible.
- \( T \): The type of the new values to replace (deduced).
- \( \text{Proj} \): The type of an optional projection function. This defaults to util::projection_identity

**Parameters**
- \( \text{policy} \): The execution policy to use for the scheduling of the iterations.
- \( \text{first} \): Refers to the beginning of the sequence of elements the algorithm will be applied to.
- \( \text{last} \): Refers to the end of the sequence of elements the algorithm will be applied to.
- \( \text{f} \): Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by \([\text{first}, \text{last})\). This is an unary predicate which returns true for the elements which need to be replaced. The signature of this predicate should be equivalent to:

```cpp
bool pred(const Type &a);
```

The signature does not need to have \&\&, but the function must not modify the objects passed to it. The type Type must be such that an object of type FwdIter can be dereferenced and then implicitly converted to Type.
- \( \text{new\_value} \): Refers to the new value to use as the replacement.
- \( \text{proj} \): Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate is invoked.

The assignments in the parallel replace_if algorithm invoked with an execution policy object of type parallel_policy or parallel_task_policy are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

**Return** The replace_if algorithm returns a hpx::future<FwdIter> if the execution policy is of type sequenced_task_policy or parallel_task_policy and returns FwdIter otherwise. It returns last.

**template** <typename ExPolicy, typename FwdIter1, typename FwdIter2, typename T1, typename T2, typename Proj = util::projection_identity>

util::detail::algorithm_result<ExPolicy, hpx::util::tagged_pair<tag::in (FwdIter1), tag::out FwdIter2>>, type replace_copyExPolicy &\&policy, FwdIter1 first, FwdIter1 last, FwdIter2 dest, T1 const &\&old_value, T2 const &\&new_value, Proj &\&proj = Proj()Copies the all elements from the range \([\text{first}, \text{last})\) to another range beginning at dest replacing all elements satisfying a specific criteria with new_value.

Effects: Assigns to every iterator \( \text{it} \) in the range \([\text{result}, \text{result} + (\text{last} - \text{first}))\) either new_value or \(*\text{it} + (\text{i} - \text{result}))\) depending on whether the following corresponding condition holds: \( \text{INVOTE}((\text{proj}, *\text{it})) \neq \text{false} \)

The assignments in the parallel replace_copy algorithm invoked with an execution policy object
of type `sequenced_policy` execute in sequential order in the calling thread.

**Note** Complexity: Performs exactly `last - first` applications of the predicate.

**Template Parameters**
- `ExPolicy`: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- `FwdIter1`: The type of the source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.
- `FwdIter2`: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an forward iterator.
- `T1`: The type of the old value to replace (deduced).
- `T2`: The type of the new values to replace (deduced).
- `Proj`: The type of an optional projection function. This defaults to `util::projection_identity`.

**Parameters**
- `policy`: The execution policy to use for the scheduling of the iterations.
- `first`: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- `last`: Refers to the end of the sequence of elements the algorithm will be applied to.
- `dest`: Refers to the beginning of the destination range.
- `old_value`: Refers to the old value of the elements to replace.
- `new_value`: Refers to the new value to use as the replacement.
- `proj`: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate is invoked.

The assignments in the parallel `replace_copy` algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

**Return** The `replace_copy` algorithm returns a `hpx::future<tagged_pair<tag::in(FwdIter1), tag::out(FwdIter2)>>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `tagged_pair<tag::in(FwdIter1), tag::out(FwdIter2)>>` otherwise.

The assignments in the parallel `replace_copy_if` algorithm invoked with an execution policy object of type `sequenced_policy` execute in sequential order in the calling thread.

**Note** Complexity: Performs exactly `last - first` applications of the predicate.

**Template Parameters**
- `ExPolicy`: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- `FwdIter1`: The type of the source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.
- `FwdIter2`: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an forward iterator.
- `F`: The type of the function/function object to use (deduced). Unlike its sequential form,
the parallel overload of `equal` requires `F` to meet the requirements of `CopyConstructible`.

• `T`: The type of the new values to replace (deduced).
• `Proj`: The type of an optional projection function. This defaults to `util::projection_identity`.

**Parameters**
- `policy`: The execution policy to use for the scheduling of the iterations.
- `first`: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- `last`: Refers to the end of the sequence of elements the algorithm will be applied to.
- `dest`: Refers to the beginning of the destination range.
- `f`: Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by `[first, last)`. This is an unary predicate which returns `true` for the elements which need to replaced. The signature of this predicate should be equivalent to:

```cpp
bool pred(const Type &a);
```

The signature does not need to have `const&`, but the function must not modify the objects passed to it. The type `Type` must be such that an object of type `FwdIter1` can be dereferenced and then implicitly converted to `Type`.

• `new_value`: Refers to the new value to use as the replacement.
• `proj`: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate is invoked.

The assignments in the parallel `replace_copy_if` algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

**Return**
The `replace_copy_if` algorithm returns a `hpx::future<tagged_pair<tag::in(FwdIter1), tag::out(FwdIter2)>>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `tagged_pair<tag::in(FwdIter1), tag::out(FwdIter2)>>` otherwise. The `replace_copy_if` algorithm returns the input iterator `last` and the output iterator to the element in the destination range, one past the last element copied.

```cpp
#include <hpx/parallel/algorithms/lexicographical_compare.hpp>

namespace hpx
{
    namespace parallel
    {
        namespace v1
        {
            template <typename ExPolicy, typename FwdIter1, typename FwdIter2, typename Pred = detail::less>
        ```
std::enable_if<execution::is_execution_policy<ExPolicy>::value, typename util::detail::algorithm_result<ExPolicy, bool>::type>::type

lexicographical_compare

Checks if the first range [first1, last1) is lexicographically less than the second range [first2, last2). uses a provided predicate to compare elements.

The comparison operations in the parallel lexicographical_compare algorithm invoked with an execution policy object of type sequenced_policy execute in sequential order in the calling thread.

**Note** Complexity: At most 2 * min(N1, N2) applications of the comparison operation, where N1 = std::distance(first1, last) and N2 = std::distance(first2, last2).

**Template Parameters**

- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- **FwdIter1**: The type of the source iterators used for the first range (deduced). This iterator type must meet the requirements of an forward iterator.
- **FwdIter2**: The type of the source iterators used for the second range (deduced). This iterator type must meet the requirements of an forward iterator.
- **Pred**: The type of an optional function/function object to use. Unlike its sequential form, the parallel overload of lexicographical_compare requires Pred to meet the requirements of CopyConstructible. This defaults to std::less<>

**Parameters**

- **policy**: The execution policy to use for the scheduling of the iterations.
- **first1**: Refers to the beginning of the sequence of elements of the first range the algorithm will be applied to.
- **last1**: Refers to the end of the sequence of elements of the first range the algorithm will be applied to.
- **first2**: Refers to the beginning of the sequence of elements of the second range the algorithm will be applied to.
- **last2**: Refers to the end of the sequence of elements of the second range the algorithm will be applied to.
- **pred**: Refers to the comparison function that the first and second ranges will be applied to.

The comparison operations in the parallel lexicographical_compare algorithm invoked with an execution policy object of type parallel_policy or parallel_task_policy are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

**Note** Lexicographical comparison is an operation with the following properties

- Two ranges are compared element by element
- The first mismatching element defines which range is lexicographically less or greater than the other
- If one range is a prefix of another, the shorter range is lexicographically less than the other
- If two ranges have equivalent elements and are of the same length, then the ranges are lexicographically equal
• An empty range is lexicographically less than any non-empty range
• Two empty ranges are lexicographically equal

Return The lexicographically_compare algorithm returns a `hpx::future<bool>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `bool` otherwise. The lexicographically_compare algorithm returns true if the first range is lexicographically less, otherwise it returns false. range `[first2, last2)`, it returns false.

#include <hpx/parallel/algorithms/uninitialized_fill.hpp>

namespace hpx

namespace parallel

namespace v1

Functions

```cpp
template <typename ExPolicy, typename FwdIter, typename T>
std::enable_if<execution::is_execution_policy<ExPolicy>::value, typename util::detail::algorithm_result<ExPolicy>::type>::type
uninitialized_fill(ExPolicy&& policy, FwdIter first, FwdIter last, T const& value)
```

Copies the given `value` to an uninitialized memory area, defined by the range `[first, last)`. If an exception is thrown during the initialization, the function has no effects.

The initializations in the parallel uninitialized_fill algorithm invoked with an execution policy object of type `sequenced_policy` execute in sequential order in the calling thread.

Note Complexity: Linear in the distance between `first` and `last`

Template Parameters

• `ExPolicy`: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
• `FwdIter`: The type of the source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.
• `T`: The type of the value to be assigned (deduced).

Parameters

• `policy`: The execution policy to use for the scheduling of the iterations.
• `first`: Refers to the beginning of the sequence of elements the algorithm will be applied to.
• `last`: Refers to the end of the sequence of elements the algorithm will be applied to.
• `value`: The value to be assigned.

The initializations in the parallel uninitialized_fill algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The uninitialized_fill algorithm returns a `hpx::future<void>`, if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns nothing otherwise.
template<typename ExPolicy, typename FwdIter, typename Size, typename T>

std::enable_if<execution::is_execution_policy<ExPolicy>::value, typename util::detail::algorithm_result<ExPolicy>::type>::type

uninitialized_fill_n(ExPolicy &&policy, FwdIter first, Size count, T const &value)

Copies the given value value to the first count elements in an uninitialized memory area beginning at first. If an exception is thrown during the initialization, the function has no effects.

The initializations in the parallel uninitialized_fill_n algorithm invoked with an execution policy object of type sequenced_policy execute in sequential order in the calling thread.

Note Complexity: Performs exactly count assignments, if count > 0, no assignments otherwise.

Template Parameters
- ExPolicy: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- FwdIter: The type of the source iterators used (deduced). This iterator type must meet the requirements of a forward iterator.
- Size: The type of the argument specifying the number of elements to apply f to.
- T: The type of the value to be assigned (deduced).

Parameters
- policy: The execution policy to use for the scheduling of the iterations.
- first: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- count: Refers to the number of elements starting at first the algorithm will be applied to.
- value: The value to be assigned.

The initializations in the parallel uninitialized_fill_n algorithm invoked with an execution policy object of type parallel_policy or parallel_task_policy are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The uninitialized_fill_n algorithm returns a hpx::future<void>, if the execution policy is of type sequenced_task_policy or parallel_task_policy and returns nothing otherwise.

#include <hpx/parallel/algorithms/includes.hpp>

namespace hpx

namespace parallel

namespace v1

Functions

template<typename ExPolicy, typename FwdIter1, typename FwdIter2, typename Pred = detail::less>
std::enable_if<execution::is_execution_policy<ExPolicy>::value, typename util::detail::algorithm_result<ExPolicy, bool>::type>::type

Returns true if every element from the sorted range [first2, last2) is found within the sorted range [first1, last1). Also returns true if [first2, last2) is empty. The version expects both ranges to be sorted with the user supplied binary predicate f.

The comparison operations in the parallel includes algorithm invoked with an execution policy object of type sequenced_policy execute in sequential order in the calling thread.

Note At most $2^*(N1+N2-1)$ comparisons, where $N1 = \text{std::distance(first1, last1)}$ and $N2 = \text{std::distance(first2, last2)}$.

Template Parameters

- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- **FwdIter1**: The type of the source iterators used for the first range (deduced). This iterator type must meet the requirements of an forward iterator.
- **FwdIter2**: The type of the source iterators used for the second range (deduced). This iterator type must meet the requirements of an forward iterator.
- **Pred**: The type of an optional function/function object to use. Unlike its sequential form, the parallel overload of includes requires Pred to meet the requirements of CopyConstructible. This defaults to std::less<>.

Parameters

- **policy**: The execution policy to use for the scheduling of the iterations.
- **first1**: Refers to the beginning of the sequence of elements of the first range the algorithm will be applied to.
- **last1**: Refers to the end of the sequence of elements of the first range the algorithm will be applied to.
- **first2**: Refers to the beginning of the sequence of elements of the second range the algorithm will be applied to.
- **last2**: Refers to the end of the sequence of elements of the second range the algorithm will be applied to.
- **op**: The binary predicate which returns true if the elements should be treated as includes. The signature of the predicate function should be equivalent to the following:

```cpp
bool pred(const Type1 &a, const Type2 &b);
```

The signature does not need to have const &, but the function must not modify the objects passed to it. The types Type1 and Type2 must be such that objects of types FwdIter1 and FwdIter2 can be dereferenced and then implicitly converted to Type1 and Type2 respectively.

The comparison operations in the parallel includes algorithm invoked with an execution policy object of type parallel_policy or parallel_task_policy are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.
Return The `includes` algorithm returns a `hpx::future<bool>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `bool` otherwise. The `includes` algorithm returns true every element from the sorted range `[first2, last2)` is found within the sorted range `[first1, last1)`. Also returns true if `[first2, last2)` is empty.

```cpp
#include <hpx/parallel/algorithms/exclusive_scan.hpp>

namespace hpx
{

namespace parallel
{

namespace v1
{

Functions

```template <typename ExPolicy, typename FwdIter1, typename FwdIter2, typename T, typename Op>
std::enable_if<execution::is_execution_policy<ExPolicy>::value, typename util::detail::algorithm_result<ExPolicy, FwdIter2>::type>::type exclusive_scan(ExPolicy&& policy, FwdIter1 first, FwdIter1 last, FwdIter2 dest, T init, Op&& op)
```

Assigns through each iterator `i` in `[result, result + (last - first))` the value of `GENERALIZED_NONCOMMUTATIVE_SUM(binary_op, init, *first, . . . , *(first + (i - result) - 1))`.

The reduce operations in the parallel `exclusive_scan` algorithm invoked with an execution policy object of type `sequenced_policy` execute in sequential order in the calling thread.

Note Complexity: `O(last - first)` applications of the predicate `op`.

Template Parameters

- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- **FwdIter1**: The type of the source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.
- **FwdIter2**: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an forward iterator.
- **T**: The type of the value to be used as initial (and intermediate) values (deduced).
- **Op**: The type of the binary function object used for the reduction operation.

Parameters

- **policy**: The execution policy to use for the scheduling of the iterations.
- **first**: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- **last**: Refers to the end of the sequence of elements the algorithm will be applied to.
- **dest**: Refers to the beginning of the destination range.
- **init**: The initial value for the generalized sum.
• **op**: Specifies the function (or function object) which will be invoked for each of the values of the input sequence. This is a binary predicate. The signature of this predicate should be equivalent to:

```cpp
Ret fun(const Type1 &a, const Type1 &b);
```

The signature does not need to have `const&`, but the function must not modify the objects passed to it. The types `Type1` and `Ret` must be such that an object of a type as given by the input sequence can be implicitly converted to any of those types.

The reduce operations in the parallel `exclusive_scan` algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

The difference between `exclusive_scan` and `inclusive_scan` is that `inclusive_scan` includes the `ith` input element in the `ith` sum. If `op` is not mathematically associative, the behavior of `inclusive_scan` may be non-deterministic.

**Return** The `copy_n` algorithm returns a `hpx::future<FwdIter2>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `FwdIter2` otherwise. The `exclusive_scan` algorithm returns the output iterator to the element in the destination range, one past the last element copied.

**Note** `GENERALIZED_NONCOMMUTATIVE_SUM(op, a1, . . . , aN)` is defined as:

- a1 when N is 1
- op(GENERALIZED_NONCOMMUTATIVE_SUM(op, a1, . . . , aK), GENERALIZED_NONCOMMUTATIVE_SUM(op, aM, . . . , aN)) where 1 < K+1 = M <= N.

```cpp
template<typename ExPolicy, typename FwdIter1, typename FwdIter2, typename T>
std::enable_if<std::is_execution_policy<ExPolicy>::value, typename util::detail::algorithm_result<ExPolicy, FwdIter2>::type>::type
exclusive_scan(ExPolicy&& policy, FwdIter1 first, FwdIter1 last, FwdIter2 dest, T init)
```

Assigns through each iterator `i` in `[result, result + (last - first))` the value of `GENERALIZED_NONCOMMUTATIVE_SUM(+, init, *first, . . . , *(first + (i - result) - 1))`

The reduce operations in the parallel `exclusive_scan` algorithm invoked with an execution policy object of type `sequenced_policy` execute in sequential order in the calling thread.

**Note** Complexity: O(last - first) applications of the predicate `std::plus<T>`.

**Template Parameters**

- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- **FwdIter1**: The type of the source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.
- **FwdIter2**: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an forward iterator.
- **T**: The type of the value to be used as initial (and intermediate) values (deduced).

**Parameters**

- **policy**: The execution policy to use for the scheduling of the iterations.
- **first**: Refers to the beginning of the sequence of elements the algorithm will be applied to.
• last: Refers to the end of the sequence of elements the algorithm will be applied to.
• dest: Refers to the beginning of the destination range.
• init: The initial value for the generalized sum.

The reduce operations in the parallel exclusive_scan algorithm invoked with an execution policy object of type parallel_policy or parallel_task_policy are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

The difference between exclusive_scan and inclusive_scan is that inclusive_scan includes the ith input element in the ith sum.

Return The copy_n algorithm returns a hpx::future<FwdIter2> if the execution policy is of type sequenced_task_policy or parallel_task_policy and returns FwdIter2 otherwise. The exclusive_scan algorithm returns the output iterator to the element in the destination range, one past the last element copied.

Note GENERALIZED_NONCOMMUTATIVE_SUM(+, a1, . . . , aN) is defined as:
• a1 when N is 1
• GENERALIZED_NONCOMMUTATIVE_SUM(+, a1, . . . , aK)
  – GENERALIZED_NONCOMMUTATIVE_SUM(+, aM, . . . , aN) where 1 < K+1 = M <= N.

#include <hpx/parallel/algorithms/move.hpp>

namespace hpx

namespace parallel

namespace v1

Functions

template <typename ExPolicy, typename FwdIter1, typename FwdIter2>
util::detail::algorithm_result<ExPolicy, hpx::util::tagged_pair<tag::in<FwdIter1>, tag::out<FwdIter2>>>::type move
  ExPolicy &&policy, FwdIter1 first, FwdIter1 last, FwdIter2 dest
Moves the elements in the range [first, last), to another range beginning at dest. After this operation the elements in the moved-from range will still contain valid values of the appropriate type, but not necessarily the same values as before the move.

The move assignments in the parallel move algorithm invoked with an execution policy object of type sequenced_policy execute in sequential order in the calling thread.

Note Complexity: Performs exactly last - first move assignments.

Template Parameters
• ExPolicy: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the move assignments.
• FwdIter1: The type of the source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.
• FwdIter2: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an forward iterator.

Parameters
• policy: The execution policy to use for the scheduling of the iterations.
• first: Refers to the beginning of the sequence of elements the algorithm will be applied to.
• last: Refers to the end of the sequence of elements the algorithm will be applied to.
• dest: Refers to the beginning of the destination range.
The move assignments in the parallel move algorithm invoked with an execution policy object of type parallel_policy or parallel_task_policy are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

**Return** The move algorithm returns a hpx::future<tagged_pair<tag::in(FwdIter1), tag::out(FwdIter2)>> if the execution policy is of type sequenced_task_policy and parallel_task_policy and returns tagged_pair<tag::in(FwdIter1), tag::out(FwdIter2)>> otherwise. The move algorithm returns the pair of the input iterator last and the output iterator to the element in the destination range, one past the last element moved.

```cpp
#include <hpx/parallel/algorithms/copy.hpp>

namespace hpx

namespace parallel

namespace v1

Functions

template <typename ExPolicy, typename FwdIter1, typename FwdIter2>
util::detail::algorithm_result<ExPolicy, hpx::util::tagged_pair<tag::in(FwdIter1), tag::out(FwdIter2)>> copy(ExPolicy &&policy, FwdIter1 first, FwdIter1 last, FwdIter2 dest)

Copies the elements in the range, defined by [first, last), to another range beginning at dest.

The assignments in the parallel copy algorithm invoked with an execution policy object of type sequenced_policy execute in sequential order in the calling thread.

**Note** Complexity: Performs exactly last - first assignments.

**Template Parameters**
- ExPolicy: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- FwdIter1: The type of the source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.
- FwdIter2: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an forward iterator.

**Parameters**
- policy: The execution policy to use for the scheduling of the iterations.
- first: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- last: Refers to the end of the sequence of elements the algorithm will be applied to.
- dest: Refers to the beginning of the destination range.

The assignments in the parallel copy algorithm invoked with an execution policy object of type parallel_policy or parallel_task_policy are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

**Return** The copy algorithm returns a hpx::future<tagged_pair<tag::in(FwdIter1), tag::out(FwdIter2)>> if the execution policy is of type sequenced_task_policy or parallel_task_policy and returns tagged_pair<tag::in(FwdIter1), tag::out(FwdIter2)>> otherwise. The copy algorithm returns the pair of the input iterator last and the output iterator to the element in the destination range, one past the last element copied.

```cpp
template <typename ExPolicy, typename FwdIter1, typename Size, typename FwdIter2>
util::detail::algorithm_result<ExPolicy, hpx::util::tagged_pair<tag::in(FwdIter1), tag::out(FwdIter2)>> copy_n(ExPolicy &&policy, FwdIter1 first, Size count, FwdIter2 dest)

Copies the count number of elements in the range, defined by [first, first + count), to another range beginning at dest.
the elements in the range [first, first + count), starting from first and proceeding to first + count - 1., to another range beginning at dest.

The assignments in the parallel `copy_n` algorithm invoked with an execution policy object of type `sequenced_policy` execute in sequential order in the calling thread.

**Note** Complexity: Performs exactly `count` assignments, if `count` > 0, no assignments otherwise.

**Template Parameters**
- `ExPolicy`: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- `FwdIter1`: The type of the source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.
- `Size`: The type of the argument specifying the number of elements to apply `f` to.
- `FwdIter2`: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an forward iterator.

**Parameters**
- `policy`: The execution policy to use for the scheduling of the iterations.
- `first`: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- `count`: Refers to the number of elements starting at `first` the algorithm will be applied to.
- `dest`: Refers to the beginning of the destination range.

The assignments in the parallel `copy_n` algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

**Return** The `copy_n` algorithm returns a `hpx::future<tagged_pair<tag::in(FwdIter1), tag::out(FwdIter2)>>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `tagged_pair<tag::in(FwdIter1), tag::out(FwdIter2)>>` otherwise. The `copy` algorithm returns the pair of the input iterator forwarded to the first element after the last in the input sequence and the output iterator to the element in the destination range, one past the last element copied.

```cpp
template <typename ExPolicy, typename FwdIter1, typename FwdIter2, typename F, typename Proj = util::projection_identity>
util::detail::algorithm_result<ExPolicy, hpx::util::tagged_pair<tag::in(FwdIter1), tag::out(FwdIter2)>> copy_if(ExPolicy && policy, FwdIter1 first, FwdIter1 last, FwdIter2 dest, F && f, Proj && proj = Proj())

Copies the elements in the range, defined by `[first, last)`, to another range beginning at `dest`. Copies only the elements for which the predicate `f` returns true. The order of the elements that are not removed is preserved.

The assignments in the parallel `copy_if` algorithm invoked with an execution policy object of type `sequenced_policy` execute in sequential order in the calling thread.

**Note** Complexity: Performs not more than `last - first` assignments, exactly `last - first` applications of the predicate `f`.

**Template Parameters**
- `ExPolicy`: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- `FwdIter1`: The type of the source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.
- `FwdIter2`: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an forward iterator.
- `F`: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of `copy_if` requires `F` to meet the requirements of `CopyConstructible`.
- `Proj`: The type of an optional projection function. This defaults to `util::projection_identity`.

**Parameters**
- `policy`: The execution policy to use for the scheduling of the iterations.
• **first**: Refers to the beginning of the sequence of elements the algorithm will be applied to.
• **last**: Refers to the end of the sequence of elements the algorithm will be applied to.
• **dest**: Refers to the beginning of the destination range.
• **f**: Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by [first, last). This is an unary predicate which returns `true` for the required elements. The signature of this predicate should be equivalent to:

```cpp
bool pred(const Type &a);
```

The signature does not need to have `const&`, but the function must not modify the objects passed to it. The type `Type` must be such that an object of type `FwdIter1` can be dereferenced and then implicitly converted to `Type`.

• **proj**: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate is invoked.

The assignments in the parallel `copy_if` algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

**Return**

The `copy_if` algorithm returns a `hpx::future<tagged_pair<tag::in(FwdIter1), tag::out(FwdIter2)>>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `tagged_pair<tag::in(FwdIter1), tag::out(FwdIter2)>>` otherwise. The `copy` algorithm returns the pair of the input iterator forwarded to the first element after the last in the input sequence and the output iterator to the element in the destination range, one past the last element copied.

```cpp
#include <hpx/parallel/algorithms/for_loop_induction.hpp>

namespace hpx
{

namespace parallel
{

namespace v2
{

Functions

template <typename T>
detail::induction_stride_helper<T> induction(T &&value, std::size_t stride)

The function template returns an induction object of unspecified type having a value type and encapsulating an initial value `value` of that type and, optionally, a stride.

For each element in the input range, a looping algorithm over input sequence `S` computes an induction value from an induction variable and ordinal position `p` within `S` by the formula `i + p * stride` if a stride was specified or `i + p` otherwise. This induction value is passed to the element access function.

If the `value` argument to `induction` is a non-const lvalue, then that lvalue becomes the live-out object for the returned induction object. For each induction object that has a live-out object, the looping algorithm assigns the value of `i + n * stride` to the live-out object upon return, where `n` is the number of elements in the input range.

**Return** This returns an induction object with value type `T`, initial value `value`, and (if specified) stride `stride`. If `T` is an lvalue of non-const type, `value` is used as the live-out object for the induction object; otherwise there is no live-out object.

**Template Parameters**
• \( T \): The value type to be used by the induction object.

**Parameters**
• \( \text{value} \): [in] The initial value to use for the induction object
• \( \text{stride} \): [in] The (optional) stride to use for the induction object (default: 1)

```cpp
#include <hpx/parallel/algorithms/uninitialized_copy.hpp>

namespace hpx
{
    namespace parallel
    {
        namespace v1
        {
            Functions
            template <typename ExPolicy, typename FwdIter1, typename FwdIter2>
            std::enable_if<execution::is_execution_policy<ExPolicy>::value, typename util::detail::algorithm_result<ExPolicy, FwdIter2>::type>::type
            uninitialized_copy(
                ExPolicy &&policy,
                FwdIter1 first,
                FwdIter1 last,
                FwdIter2 dest)
            {
                // Implementation...
            }
        }
    }
}
```

Copies the elements in the range, defined by \([\text{first}, \text{last})\), to an uninitialized memory area beginning at \( \text{dest} \). If an exception is thrown during the copy operation, the function has no effects.

The assignments in the parallel \text{uninitialized_copy} algorithm invoked with an execution policy object of type \text{sequenced_policy} execute in sequential order in the calling thread.

**Note** Complexity: Performs exactly \( \text{last} - \text{first} \) assignments.

**Template Parameters**
• \( \text{ExPolicy} \): The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
• \( \text{FwdIter1} \): The type of the source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.
• \( \text{FwdIter2} \): The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of a forward iterator.

**Parameters**
• \( \text{policy} \): The execution policy to use for the scheduling of the iterations.
• \( \text{first} \): Refers to the beginning of the sequence of elements the algorithm will be applied to.
• \( \text{last} \): Refers to the end of the sequence of elements the algorithm will be applied to.
• \( \text{dest} \): Refers to the beginning of the destination range.

The assignments in the parallel \text{uninitialized_copy} algorithm invoked with an execution policy object of type \text{parallel_policy} or \text{parallel_task_policy} are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

**Return** The \text{uninitialized_copy} algorithm returns a \text{hpx::future<\text{FwdIter2}>}, if the execution policy is of type \text{sequenced_task_policy} or \text{parallel_task_policy} and returns \text{FwdIter2} otherwise. The \text{uninitialized_copy} algorithm returns the output iterator to the element in the destination range, one past the last element copied.
template <typename ExPolicy, typename FwdIter1, typename Size, typename FwdIter2>
std::enable_if<execution::is_execution_policy<ExPolicy>::value, typename util::detail::algorithm_result<ExPolicy, FwdIter2>::type>::type
uninitialized_copy_n(
    ExPolicy&& policy, FwdIter1 first, Size count, FwdIter2 dest)

Copies the elements in the range [first, first + count), starting from first and proceeding to first + count - 1., to another range beginning at dest. If an exception is thrown during the copy operation, the function has no effects.

The assignments in the parallel uninitialized_copy_n algorithm invoked with an execution policy object of type sequenced_policy execute in sequential order in the calling thread.

**Note** Complexity: Performs exactly count assignments, if count > 0, no assignments otherwise.

**Template Parameters**
- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- **FwdIter1**: The type of the source iterators used (deduced). This iterator type must meet the requirements of an input iterator.
- **Size**: The type of the argument specifying the number of elements to apply f to.
- **FwdIter2**: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of a forward iterator.

**Parameters**
- **policy**: The execution policy to use for the scheduling of the iterations.
- **first**: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- **count**: Refers to the number of elements starting at first the algorithm will be applied to.
- **dest**: Refers to the beginning of the destination range.

The assignments in the parallel uninitialized_copy_n algorithm invoked with an execution policy object of type parallel_policy or parallel_task_policy are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

**Return** The uninitialized_copy_n algorithm returns a hpx::future<FwdIter2> if the execution policy is of type sequenced_task_policy or parallel_task_policy and returns FwdIter2 otherwise. The uninitialized_copy_n algorithm returns the output iterator to the element in the destination range, one past the last element copied.

```cpp
#include <hpx/parallel/algorithms/generate.hpp>

namespace hpx

namespace parallel

namespace v1

Functions

template <typename ExPolicy, typename FwdIter, typename F>
```
Assign each element in range \([\text{first}, \text{last})\) a value generated by the given function object \(f\).

The assignments in the parallel \textit{generate} algorithm invoked with an execution policy object of type \textit{sequenced\_policy} execute in sequential order in the calling thread.

\textbf{Note} Complexity: Exactly \(\text{distance(\text{first}, \text{last})}\) invocations of \(f\) and assignments.

\textbf{Template Parameters}

- \texttt{ExPolicy}: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- \texttt{FwdIter}: The type of the source iterators used (deduced). This iterator type must meet the requirements of a forward iterator.
- \texttt{F}: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of \textit{equal} requires \(F\) to meet the requirements of \textit{CopyConstructible}.

\textbf{Parameters}

- \texttt{policy}: The execution policy to use for the scheduling of the iterations.
- \texttt{first}: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- \texttt{last}: Refers to the end of the sequence of elements the algorithm will be applied to.
- \texttt{f}: generator function that will be called. signature of function should be equivalent to the following:

\[
\text{Ret fun();}
\]

The type \(\text{Ret}\) must be such that an object of type \texttt{FwdIter} can be dereferenced and assigned a value of type \(\text{Ret}\).

The assignments in the parallel \textit{generate} algorithm invoked with an execution policy object of type \textit{parallel\_policy} or \textit{parallel\_task\_policy} are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

\textbf{Return} The \textit{replace\_if} algorithm returns a \texttt{hpx::future<FwdIter>} if the execution policy is of type \textit{sequenced\_task\_policy} or \textit{parallel\_task\_policy} and returns \texttt{FwdIter} otherwise. It returns \texttt{last}.

\begin{verbatim}
template <typename ExPolicy, typename FwdIter, typename Size, typename F>
util::detail::algorithm_result<ExPolicy, FwdIter>::type
    generate_n (ExPolicy &&policy,
        FwdIter first, Size count,
        F &&f)
\end{verbatim}

Assigns each element in range \([\text{first}, \text{first+count})\) a value generated by the given function object \(g\).

The assignments in the parallel \textit{generate\_n} algorithm invoked with an execution policy object of type \textit{sequenced\_policy} execute in sequential order in the calling thread.

\textbf{Note} Complexity: Exactly \(\text{count}\) invocations of \(f\) and assignments, for \(\text{count} > 0\).

\textbf{Template Parameters}

- \texttt{ExPolicy}: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- \texttt{FwdIter}: The type of the source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.
- \texttt{F}: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of \textit{equal} requires \(F\) to meet the requirements of \textit{CopyConstructible}.

\textbf{Parameters}

- \texttt{policy}: The execution policy to use for the scheduling of the iterations.
• **first**: Refers to the beginning of the sequence of elements the algorithm will be applied to.
• **count**: Refers to the number of elements in the sequence the algorithm will be applied to.
• **f**: Refers to the generator function object that will be called. The signature of the function should be equivalent to

```
Ret fun();
```

The type `Ret` must be such that an object of type `OutputIt` can be dereferenced and assigned a value of type `Ret`.

The assignments in the parallel `generate_n` algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

**Return** The `replace_if` algorithm returns a `hpx::future<FwdIter>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `FwdIter` otherwise. It returns `last`.

```cpp
#include <hpx/parallel/algorithms/inclusive_scan.hpp>

namespace hpx
{
  namespace parallel
  {
    namespace v1
    {
      Functions

      template <typename ExPolicy, typename FwdIter1, typename FwdIter2, typename Op, typename T>
      util::detail::algorithm_result<ExPolicy, FwdIter2>::type inclusive_scan(ExPolicy && policy, FwdIter1 first, FwdIter1 last, FwdIter2 dest, Op && op, T init)
      {
        Assigns through each iterator `i` in `[result, result + (last - first))` the value of `GENERALIZED_NONCOMMUTATIVE_SUM(op, init, *first, ..., *(first + (i - result)))`.

        The reduce operations in the `inclusive_scan` algorithm invoked with an execution policy object of type `sequenced_policy` execute in sequential order in the calling thread.

        **Note** Complexity: O(last - first) applications of the predicate `op`.

        **Template Parameters**
        • `ExPolicy`: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
        • `FwdIter1`: The type of the source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.
        • `FwdIter2`: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an forward iterator.
        • `T`: The type of the value to be used as initial (and intermediate) values (deduced).
        • `Op`: The type of the binary function object used for the reduction operation.

        **Parameters**
        • `policy`: The execution policy to use for the scheduling of the iterations.
```
• first: Refers to the beginning of the sequence of elements the algorithm will be applied to.
• last: Refers to the end of the sequence of elements the algorithm will be applied to.
• dest: Refers to the beginning of the destination range.
• init: The initial value for the generalized sum.
• op: Specifies the function (or function object) which will be invoked for each of the values of the input sequence. This is a binary predicate. The signature of this predicate should be equivalent to:

\[
\text{Ret fun}(\text{const Type1 &a, const Type1 &b});
\]

The signature does not need to have const& but the function must not modify the objects passed to it. The types Type1 and Ret must be such that an object of a type as given by the input sequence can be implicitly converted to any of those types.

The reduce operations in the parallel inclusive_scan algorithm invoked with an execution policy object of type parallel_policy or parallel_task_policy are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

The difference between exclusive_scan and inclusive_scan is that inclusive_scan includes the ith input element in the ith sum. If op is not mathematically associative, the behavior of inclusive_scan may be non-deterministic.

Return The copy_n algorithm returns a hpx::future<FwdIter2> if the execution policy is of type sequenced_task_policy or parallel_task_policy and returns FwdIter2 otherwise. The inclusive_scan algorithm returns the output iterator to the element in the destination range, one past the last element copied.

Note GENERALIZED_NONCOMMUTATIVE_SUM(op, a1, . . . , aN) is defined as:

• a1 when N is 1
• op(GENERALIZED_NONCOMMUTATIVE_SUM(op, a1, . . . , aK), GENERALIZED_NONCOMMUTATIVE_SUM(op, aM, . . . , aN)) where 1 < K+1 = M <= N.

\[
\text{template } \langle \text{typename } \text{ExPolicy, typename } \text{FwdIter1, typename } \text{FwdIter2, typename } \text{Op}\rangle \\
\text{util::detail::algorithm_result<ExPolicy, FwdIter2>::type inclusive_scan} \langle \text{ExPolicy } &\&\text{policy}, \text{FwdIter1} \text{ first}, \text{FwdIter1} \text{ last}, \text{FwdIter2} \text{ dest}, \text{Op } &\&\text{op}\rangle
\]

Assigns through each iterator i in \{result, result + (last - first)) the value of GENERALIZED_NONCOMMUTATIVE_SUM(op, *first, . . . , *(first + (i - result))).

The reduce operations in the parallel inclusive_scan algorithm invoked with an execution policy object of type sequenced_policy execute in sequential order in the calling thread.

Note Complexity: O(last - first) applications of the predicate op.

Template Parameters
• ExPolicy: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
• FwdIter1: The type of the source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.
• FwdIter2: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an forward iterator.
• Op: The type of the binary function object used for the reduction operation.

Parameters
• policy: The execution policy to use for the scheduling of the iterations.
• first: Refers to the beginning of the sequence of elements the algorithm will be applied to.
• last: Refers to the end of the sequence of elements the algorithm will be applied to.
• dest: Refers to the beginning of the destination range.
• op: Specifies the function (or function object) which will be invoked for each of the values of the input sequence. This is a binary predicate. The signature of this predicate should be equivalent to:

```cpp
Ret fun(const Type1 &a, const Type1 &b);
```

The signature does not need to have const&, but the function must not modify the objects passed to it. The types Type1 and Ret must be such that an object of a type as given by the input sequence can be implicitly converted to any of those types.

The reduce operations in the parallel `inclusive_scan` algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

The difference between `exclusive_scan` and `inclusive_scan` is that `inclusive_scan` includes the ith input element in the ith sum.

**Return** The `copy_n` algorithm returns a `hpx::future<FwdIter2>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `FwdIter2` otherwise. The `inclusive_scan` algorithm returns the output iterator to the element in the destination range, one past the last element copied.

**Note** GENERALIZED_NONCOMMUTATIVE_SUM(+, a1, . . . , aN) is defined as:
- a1 when N is 1
- GENERALIZED_NONCOMMUTATIVE_SUM(op, a1, . . . , aK)
  - GENERALIZED_NONCOMMUTATIVE_SUM(+, aM, . . . , aN) where 1 < K+1 = M <= N.

```cpp
template <typename ExPolicy, typename FwdIter1, typename FwdIter2>
std::enable_if<execution::is_execution_policy<ExPolicy>::value, typename util::detail::algorithm_result<ExPolicy, FwdIter2>::type>::type
inclusive_scan(ExPolicy &&policy, FwdIter1 first, FwdIter1 last, FwdIter2 dest)
```

Assigns through each iterator i in [result, result + (last - first)) the value of GENERALIZED_NONCOMMUTATIVE_SUM(+, *first, . . . , *(first + (i - result))).

The reduce operations in the parallel `inclusive_scan` algorithm invoked with an execution policy object of type `sequenced_policy` execute in sequential order in the calling thread.

**Note** Complexity: O(last - first) applications of the predicate op.

**Template Parameters**
- ExPolicy: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- FwdIter1: The type of the source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.
- FwdIter2: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an forward iterator.

**Parameters**
- policy: The execution policy to use for the scheduling of the iterations.
- first: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- last: Refers to the end of the sequence of elements the algorithm will be applied to.
- dest: Refers to the beginning of the destination range.
The reduce operations in the parallel *inclusive_scan* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

The difference between *exclusive_scan* and *inclusive_scan* is that *inclusive_scan* includes the ith input element in the ith sum.

**Return** The *copy_n* algorithm returns a *hpx::future<FwdIter2>* if the execution policy is of type *sequenced_task_policy* or *parallel_task_policy* and returns *FwdIter2* otherwise. The *inclusive_scan* algorithm returns the output iterator to the element in the destination range, one past the last element copied.

**Note** GENERALIZED_NONCOMMUTATIVE_SUM(+, a1, ..., aN) is defined as:
- a1 when N is 1
- GENERALIZED_NONCOMMUTATIVE_SUM(+, a1, ..., aK) - GENERALIZED_NONCOMMUTATIVE_SUM(+, aM, ..., aN) where 1 < K+1 = M <= N.

```
#include <hpx/parallel/algorithms/mismatch.hpp>

namespace hpx
{
  namespace parallel
  {
    namespace v1
    {

      Functions

      template <typename ExPolicy, typename FwdIter1, typename FwdIter2, typename Pred = detail::equal_to>
      std::enable_if<execution::is_execution_policy<ExPolicy>::value, typename util::detail::algorithm_result<ExPolicy, std::pair<FwdIter1, FwdIter2>>>::type
      mismatch(ExPolicy &&policy, FwdIter1 first1, FwdIter1 last1, FwdIter2 first2, FwdIter2 last2, Pred &&op = Pred())
      {
        Returns true if the range [first1, last1) is mismatch to the range [first2, last2), and false otherwise.
        The comparison operations in the parallel *mismatch* algorithm invoked with an execution policy object of type *sequenced_policy* execute in sequential order in the calling thread.
        **Note** Complexity: At most min(last1 - first1, last2 - first2) applications of the predicate *f*. If *FwdIter1* and *FwdIter2* meet the requirements of *RandomAccessIterator* and (last1 - first1) != (last2 - first2) then no applications of the predicate *f* are made.

      Template Parameters
      • ExPolicy: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
```
• **FwdIter1**: The type of the source iterators used for the first range (deduced). This iterator type must meet the requirements of an forward iterator.
• **FwdIter2**: The type of the source iterators used for the second range (deduced). This iterator type must meet the requirements of an forward iterator.
• **Pred**: The type of an optional function/function object to use. Unlike its sequential form, the parallel overload of **mismatch** requires **Pred** to meet the requirements of **CopyConstructible**. This defaults to std::equal_to<>.

### Parameters
- **policy**: The execution policy to use for the scheduling of the iterations.
- **first1**: Refers to the beginning of the sequence of elements of the first range the algorithm will be applied to.
- **last1**: Refers to the end of the sequence of elements of the first range the algorithm will be applied to.
- **first2**: Refers to the beginning of the sequence of elements of the second range the algorithm will be applied to.
- **last2**: Refers to the end of the sequence of elements of the second range the algorithm will be applied to.
- **op**: The binary predicate which returns true if the elements should be treated as mismatch. The signature of the predicate function should be equivalent to the following:

```cpp
bool pred(const Type1 &a, const Type2 &b);
```

The signature does not need to have const &, but the function must not modify the objects passed to it. The types Type1 and Type2 must be such that objects of types FwdIter1 and FwdIter2 can be dereferenced and then implicitly converted to Type1 and Type2 respectively.

The comparison operations in the parallel **mismatch** algorithm invoked with an execution policy object of type parallel_policy or parallel_task_policy are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

### Note
The two ranges are considered mismatch if, for every iterator i in the range [first1,last1), *(i mismatchs *(first2 + (i - first1))). This overload of mismatch uses operator== to determine if two elements are mismatch.

### Return
The **mismatch** algorithm returns a hpx::future<bool> if the execution policy is of type sequenced_task_policy or parallel_task_policy and returns bool otherwise. The mismatch algorithm returns true if the elements in the two ranges are mismatch, otherwise it returns false. If the length of the range [first1, last1) does not mismatch the length of the range [first2, last2), it returns false.

```cpp
template<typename ExPolicy, typename FwdIter1, typename FwdIter2, typename Pred = detail::equal_to>
std::enable_if<execution::is_execution_policy<ExPolicy>::value, typename util::detail::algorithm_result<ExPolicy, std::pair<FwdIter1, FwdIter2>>>::type
mismatch(ExPolicy &&policy, FwdIter1 first1, FwdIter1 last1, FwdIter2 first2, Pred &&op = Pred());
```

Returns std::pair with iterators to the first two non-equivalent elements.

The comparison operations in the parallel **mismatch** algorithm invoked with an execution policy object of type sequenced_policy execute in sequential order in the calling thread.
Note  Complexity: At most last1 - first1 applications of the predicate f.

Template Parameters

- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- **FwdIter1**: The type of the source iterators used for the first range (deduced). This iterator type must meet the requirements of an forward iterator.
- **FwdIter2**: The type of the source iterators used for the second range (deduced). This iterator type must meet the requirements of an forward iterator.
- **Pred**: The type of an optional function/function object to use. Unlike its sequential form, the parallel overload of *mismatch* requires *Pred* to meet the requirements of *CopyConstructible*. This defaults to std::equal_to<>

Parameters

- **policy**: The execution policy to use for the scheduling of the iterations.
- **first1**: Refers to the beginning of the sequence of elements of the first range the algorithm will be applied to.
- **last1**: Refers to the end of the sequence of elements of the first range the algorithm will be applied to.
- **first2**: Refers to the beginning of the sequence of elements of the second range the algorithm will be applied to.
- **op**: The binary predicate which returns true if the elements should be treated as mismatch. The signature of the predicate function should be equivalent to the following:

```cpp
bool pred(const Type1 &a, const Type2 &b);
```

The signature does not need to have const &, but the function must not modify the objects passed to it. The types Type1 and Type2 must be such that objects of types FwdIter1 and FwdIter2 can be dereferenced and then implicitly converted to Type1 and Type2 respectively.

The comparison operations in the parallel *mismatch* algorithm invoked with an execution policy object of type *parallel_policy* or *parallel_task_policy* are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Return The *mismatch* algorithm returns a *hpx::future<std::pair<FwdIter1, FwdIter2>>* if the execution policy is of type *sequenced_task_policy* or *parallel_task_policy* and returns *std::pair<FwdIter1, FwdIter2>* otherwise. The *mismatch* algorithm returns the first mismatching pair of elements from two ranges: one defined by [first1, last1) and another defined by [first2, last2).

```cpp
#include <hpx/parallel/algorithms/set_symmetric_difference.hpp>

namespace hpx

namespace parallel

namespace v1

Functions

template <typename ExPolicy, typename FwdIter1, typename FwdIter2, typename FwdIter3, typename...>
std::enable_if<execution::is_execution_policy<ExPolicy>::value, typename util::detail::algorithm_result<ExPolicy, FwdIter3>::type>::type

Constructs a sorted range beginning at dest consisting of all elements present in either of the sorted ranges \([\text{first1}, \text{last1})\) and \([\text{first2}, \text{last2}),\) but not in both of them are copied to the range beginning at \(dest.\) The resulting range is also sorted. This algorithm expects both input ranges to be sorted with the given binary predicate \(f.\)

If some element is found \(m\) times in \([\text{first1}, \text{last1})\) and \(n\) times in \([\text{first2}, \text{last2}),\) it will be copied to \(dest\) exactly \(std::abs(m-n)\) times. If \(m > n,\) then the last \(m-n\) of those elements are copied from \([\text{first1},\text{last1}),\) otherwise the last \(n-m\) elements are copied from \([\text{first2},\text{last2}).\) The resulting range cannot overlap with either of the input ranges.

**Note** Complexity: At most \(2*(N_1 + N_2 - 1)\) comparisons, where \(N_1\) is the length of the first sequence and \(N_2\) is the length of the second sequence.
The resulting range cannot overlap with either of the input ranges.

The application of function objects in parallel algorithm invoked with a sequential execution policy object execute in sequential order in the calling thread (\(sequenced_policy\)) or in a single new thread spawned from the current thread (for \(sequenced_task_policy\)).

**Template Parameters**
- \(\text{ExPolicy}\): The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it applies user-provided function objects.
- \(\text{FwdIter1}\): The type of the source iterators used (deduced) representing the first sequence. This iterator type must meet the requirements of an forward iterator.
- \(\text{FwdIter2}\): The type of the source iterators used (deduced) representing the first sequence. This iterator type must meet the requirements of an forward iterator.
- \(\text{FwdIter3}\): The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an output iterator.
- \(\text{Pred}\): The type of an optional function/function object to use. Unlike its sequential form, the parallel overload of \(set\_symmetric\_difference\) requires \(\text{Pred}\) to meet the requirements of \(\text{CopyConstructible.}\) This defaults to \(\text{std::less<}\)

**Parameters**
- \(\text{policy}\): The execution policy to use for the scheduling of the iterations.
- \(\text{first1}\): Refers to the beginning of the sequence of elements of the first range the algorithm will be applied to.
- \(\text{last1}\): Refers to the end of the sequence of elements of the first range the algorithm will be applied to.
- \(\text{first2}\): Refers to the beginning of the sequence of elements of the second range the algorithm will be applied to.
- \(\text{last2}\): Refers to the end of the sequence of elements of the second range the algorithm will be applied to.
will be applied to.

- **dest**: Refers to the beginning of the destination range.
- **op**: The binary predicate which returns true if the elements should be treated as equal. The signature of the predicate function should be equivalent to the following:

```cpp
bool pred(const Type1 &a, const Type1 &b);
```

The signature does not need to have const &, but the function must not modify the objects passed to it. The type `Type1` must be such that objects of type `InIter` can be dereferenced and then implicitly converted to `Type1`.

The application of function objects in parallel algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

**Return**

The `set_symmetric_difference` algorithm returns a `hpx::future<FwdIter3>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `FwdIter3` otherwise. The `set_symmetric_difference` algorithm returns the output iterator to the element in the destination range, one past the last element copied.

```cpp
#include <hpx/parallel/algorithms/is_heap.hpp>

namespace hpx
{
namespace parallel
{
namespace v1
{

Functions

```cpp
template <typename ExPolicy, typename RandIter, typename Comp = detail::less, typename Proj = util::projection_identity>
util::detail::algorithm_result<ExPolicy, bool>::type is_heap(ExPolicy &&policy, RandIter first, RandIter last, Comp &&comp = Comp(), Proj &&proj = Proj())
```

Returns whether the range is max heap. That is, true if the range is max heap, false otherwise. The function uses the given comparison function object `comp` (defaults to using `operator<()`).

**comp** has to induce a strict weak ordering on the values.

**Note**  Complexity: Performs at most N applications of the comparison `comp`, at most 2 * N applications of the projection `proj`, where N = last - first.

**Template Parameters**

- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- **RandIter**: The type of the source iterators used (deduced). This iterator type must meet the requirements of a random access iterator.
- **Comp**: The type of the function/function object to use (deduced).
- **Proj**: The type of an optional projection function. This defaults to `util::projection_identity`.

**Parameters**

- **policy**: The execution policy to use for the scheduling of the iterations.
- **first**: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- **last**: Refers to the end of the sequence of elements the algorithm will be applied to.
• **comp**: `comp` is a callable object. The return value of the INVOKE operation applied to an object of type `Comp`, when contextually converted to bool, yields true if the first argument of the call is less than the second, and false otherwise. It is assumed that `comp` will not apply any non-constant function through the dereferenced iterator.

• **proj**: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate is invoked.

The application of function objects in parallel algorithm invoked with an execution policy object of type `sequenced_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

**Return** The `is_heap` algorithm returns a `hpx::future<bool>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `bool` otherwise. The `is_heap` algorithm returns whether the range is max heap. That is, true if the range is max heap, false otherwise.

```cpp
template <typename ExPolicy, typename RandIter, typename Comp = detail::less, typename Proj = util::projection_identity>
util::detail::algorithm_result<ExPolicy, RandIter>::type is_heap_until(ExPolicy &&policy, RandIter first, RandIter last, Comp &&comp = Comp(), Proj &&proj = Proj())
```

Returns the upper bound of the largest range beginning at `first` which is a max heap. That is, the last iterator `it` for which range `[first, it)` is a max heap. The function uses the given comparison function object `comp` (defaults to using `operator<()`).

`comp` has to induce a strict weak ordering on the values.

**Note** Complexity: Performs at most `N` applications of the comparison `comp`, at most `2 * N` applications of the projection `proj`, where `N = last - first`.

**Template Parameters**

• **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.

• **RandIter**: The type of the source iterators used (deduced). This iterator type must meet the requirements of a random access iterator.

• **Comp**: The type of the function/function object to use (deduced).

• **Proj**: The type of an optional projection function. This defaults to `util::projection_identity`

**Parameters**

• **policy**: The execution policy to use for the scheduling of the iterations.

• **first**: Refers to the beginning of the sequence of elements the algorithm will be applied to.

• **last**: Refers to the end of the sequence of elements the algorithm will be applied to.

• **comp**: `comp` is a callable object. The return value of the INVOKE operation applied to an object of type `Comp`, when contextually converted to bool, yields true if the first argument of the call is less than the second, and false otherwise. It is assumed that `comp` will not apply any non-constant function through the dereferenced iterator.

• **proj**: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate is invoked.

The application of function objects in parallel algorithm invoked with an execution policy object of type `sequenced_policy` execute in sequential order in the calling thread.

The application of function objects in parallel algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.
Return The `is_heap_until` algorithm returns a `hpx::future<RandIter>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `RandIter` otherwise. The `is_heap_until` algorithm returns the upper bound of the largest range beginning at `first` which is a max heap. That is, the last iterator `it` for which range `[first, it)` is a max heap.

```cpp
#include <hpx/parallel/algorithms/equal.hpp>
namespace hpx
{
    namespace parallel
    {
        namespace v1
        {
            template <typename ExPolicy, typename FwdIter1, typename FwdIter2, typename Pred = detail::equal_to>
            std::enable_if<execution::is_execution_policy<ExPolicy>::value, typename util::detail::algorithm_result<ExPolicy, bool>::type>::type
            equal(ExPolicy &&policy, FwdIter1 first1, FwdIter1 last1, FwdIter2 first2, FwdIter2 last2, Pred &&op = Pred())
            Returns true if the range `[first1, last1)` is equal to the range `[first2, last2)`, and false otherwise.
            The comparison operations in the parallel `equal` algorithm invoked with an execution policy object of type `sequenced_policy` execute in sequential order in the calling thread.
            Note Complexity: At most \( \min(last1 - first1, last2 - first2) \) applications of the predicate \( f \).
            Template Parameters
            - `ExPolicy`: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
            - `FwdIter1`: The type of the source iterators used for the first range (deduced). This iterator type must meet the requirements of an forward iterator.
            - `FwdIter2`: The type of the source iterators used for the second range (deduced). This iterator type must meet the requirements of an forward iterator.
            - `Pred`: The type of an optional function/function object to use. Unlike its sequential form, the parallel overload of `equal` requires `Pred` to meet the requirements of `CopyConstructible`. This defaults to `std::equal_to<>`
            Parameters
            - `policy`: The execution policy to use for the scheduling of the iterations.
            - `first1`: Refers to the beginning of the sequence of elements of the first range the algorithm will be applied to.
            - `last1`: Refers to the end of the sequence of elements of the first range the algorithm will be applied to.
        }
    }
}
```
• **first2**: Refers to the beginning of the sequence of elements of the second range the algorithm will be applied to.
• **last2**: Refers to the end of the sequence of elements of the second range the algorithm will be applied to.
• **op**: The binary predicate which returns true if the elements should be treated as equal. The signature of the predicate function should be equivalent to the following:

```cpp
bool pred(const Type1 &a, const Type2 &b);
```

The signature does not need to have const &, but the function must not modify the objects passed to it. The types `Type1` and `Type2` must be such that objects of types `FwdIter1` and `FwdIter2` can be dereferenced and then implicitly converted to `Type1` and `Type2` respectively.

The comparison operations in the parallel `equal` algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

**Note**  The two ranges are considered equal if, for every iterator `i` in the range \([\text{first1}, \text{last1})\), \(^{*}i\) equals \(^{(\text{first2} + (i - \text{first1}))}\). This overload of `equal` uses operator\(==\) to determine if two elements are equal.

**Return** The `equal` algorithm returns a `hpx::future<bool>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `bool` otherwise. The `equal` algorithm returns true if the elements in the two ranges are equal, otherwise it returns false. If the length of the range \([\text{first1}, \text{last1})\) does not equal the length of the range \([\text{first2}, \text{last2})\), it returns false.

```cpp
template <typename ExPolicy, typename FwdIter1, typename FwdIter2, typename Pred = detail::equal_to>
std::enable_if<
    execution::is_execution_policy<ExPolicy>::value, typename util::detail::algorithm_result<ExPolicy, bool>::type>::type
equal(
    ExPolicy &&policy,
    FwdIter1 first1,
    FwdIter1 last1,
    FwdIter2 first2,
    Pred &&op = Pred())
```

Returns true if the range \([\text{first1}, \text{last1})\) is equal to the range starting at `first2`, and false otherwise.

The comparison operations in the parallel `equal` algorithm invoked with an execution policy object of type `sequenced_policy` execute in sequential order in the calling thread.

**Note** Complexity: At most \(last1 - first1\) applications of the predicate `f`.

**Template Parameters**
- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- **FwdIter1**: The type of the source iterators used for the first range (deduced). This iterator type must meet the requirements of an forward iterator.
- **FwdIter2**: The type of the source iterators used for the second range (deduced). This iterator type must meet the requirements of an forward iterator.
- **Pred**: The type of an optional function/function object to use. Unlike its sequential form, the parallel overload of `equal` requires `Pred` to meet the requirements of `CopyConstructible`. This defaults to `std::equal_to<>`

**Parameters**
- **policy**: The execution policy to use for the scheduling of the iterations.
• first1: Refers to the beginning of the sequence of elements of the first range the algorithm will be applied to.
• last1: Refers to the end of the sequence of elements of the first range the algorithm will be applied to.
• first2: Refers to the beginning of the sequence of elements of the second range the algorithm will be applied to.
• op: The binary predicate which returns true if the elements should be treated as equal. The signature of the predicate function should be equivalent to the following:

```cpp
bool pred(const Type1 &a, const Type2 &b);
```

The signature does not need to have const &, but the function must not modify the objects passed to it. The types Type1 and Type2 must be such that objects of types FwdIter1 and FwdIter2 can be dereferenced and then implicitly converted to Type1 and Type2 respectively.

The comparison operations in the parallel equal algorithm invoked with an execution policy object of type parallel_policy or parallel_task_policy are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

**Note** The two ranges are considered equal if, for every iterator i in the range (first1,last1), *i equals *(first2 + (i - first1)). This overload of equal uses operator== to determine if two elements are equal.

**Return** The equal algorithm returns a hpx::future<bool> if the execution policy is of type sequenced_task_policy or parallel_task_policy and returns bool otherwise. The equal algorithm returns true if the elements in the two ranges are equal, otherwise it returns false.

```cpp
#include <hpx/parallel/algorithms/remove_copy.hpp>

namespace hpx
{
    namespace parallel
    {
        namespace v1
        {
            template <typename ExPolicy, typename FwdIter1, typename FwdIter2, typename T, typename Proj = util::projection_identity>
            util::detail::algorithm_result<ExPolicy, hpx::util::tagged_pair<tag::in(FwdIter1), tag::out(FwdIter2)>> remove_copy(ExPolicy &&policy, FwdIter1 first, FwdIter1 last, FwdIter2 dest, T const &val, Proj &&proj = Proj())Copies the elements in the range, defined by [first, last), to another range beginning at dest. Copies only the elements for which the comparison operator returns false when compare to val. The order of the elements that are not removed is preserved.

            Effects: Copies all the elements referred to by the iterator it in the range [first,last) for which the following corresponding conditions do not hold: INVOKE(proj, *it) == value

            The assignments in the parallel remove_copy algorithm invoked with an execution policy object of type sequenced_policy execute in sequential order in the calling thread.

            **Note** Complexity: Performs not more than last - first assignments, exactly last - first applications of the predicate f.

            **Template Parameters**
            • ExPolicy: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
**FwdIter1**: The type of the source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.

**FwdIter2**: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an forward iterator.

**T**: The type that the result of dereferencing FwdIter1 is compared to.

**Proj**: The type of an optional projection function. This defaults to `util::projection_identity`

**Parameters**

- **policy**: The execution policy to use for the scheduling of the iterations.
- **first**: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- **last**: Refers to the end of the sequence of elements the algorithm will be applied to.
- **dest**: Refers to the beginning of the destination range.
- **val**: Value to be removed.
- **proj**: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate is invoked.

The assignments in the parallel `remove_copy` algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

**Return**

The `remove_copy` algorithm returns a `hpx::future<tagged_pair<tag::in(FwdIter1), tag::out(FwdIter2)>>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `tagged_pair<tag::in(FwdIter1), tag::out(FwdIter2)>>` otherwise.

The `copy` algorithm returns the pair of the input iterator forwarded to the first element after the last in the input sequence and the output iterator to the element in the destination range, one past the last element copied.

**Template Parameters**

- **ExPolicy**: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- **FwdIter1**: The type of the source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.
- **FwdIter2**: The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an forward iterator.
- **F**: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of `copy_if` requires `F` to meet the requirements of `CopyConstructible`.
- **Proj**: The type of an optional projection function. This defaults to `util::projection_identity`

**Parameters**

- **policy**: The execution policy to use for the scheduling of the iterations.
- **first**: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- **last**: Refers to the end of the sequence of elements the algorithm will be applied to.
• dest: Refers to the beginning of the destination range.
• f: Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by [first, last). This is an unary predicate which returns true for the elements to be removed. The signature of this predicate should be equivalent to:

```cpp
bool pred(const Type &a);
```

The signature does not need to have const&, but the function must not modify the objects passed to it. The type Type must be such that an object of type FwdIter1 can be dereferenced and then implicitly converted to Type.

• proj: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate is invoked.

The assignments in the parallel remove_copy_if algorithm invoked with an execution policy object of type parallel_policy or parallel_task_policy are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

**Return** The remove_copy_if algorithm returns a hpx::future<tagged_pair<tag::in(FwdIter1), tag::out(FwdIter2)>> if the execution policy is of type sequenced_task_policy or parallel_task_policy and returns tagged_pair<tag::in(FwdIter1), tag::out(FwdIter2)>> otherwise. The copy algorithm returns the pair of the input iterator forwarded to the first element after the last in the input sequence and the output iterator to the element in the destination range, one past the last element copied.

```
#include <hpx/parallel/algorithms/destroy.hpp>

namespace hpx
{

namespace parallel
{

namespace v1
{

Functions

**template** <typename ExPolicy, typename FwdIter>
util::detail::algorithm_result<ExPolicy>::type destroy(ExPolicy &policy, FwdIter first, FwdIter last)

Destroys objects of type typename iterator_traits<ForwardIt>::value_type in the range [first, last).

The operations in the parallel destroy algorithm invoked with an execution policy object of type sequenced_policy execute in sequential order in the calling thread.

**Note** Complexity: Performs exactly last - first operations.

**Template Parameters**

• ExPolicy: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.

• FwdIter: The type of the source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.

**Parameters**

• policy: The execution policy to use for the scheduling of the iterations.

• first: Refers to the beginning of the sequence of elements the algorithm will be applied to.

• last: Refers to the end of the sequence of elements the algorithm will be applied to.
The operations in the parallel `destroy` algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

**Return** The `destroy` algorithm returns a `hpx::future<void>`, if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `void` otherwise.

```cpp
template <typename ExPolicy, typename FwdIter, typename Size>
util::detail::algorithm_result<ExPolicy, FwdIter>::type destroy_n (ExPolicy && policy, FwdIter first, Size count)
```

Destroys objects of type typename iterator_traits<ForwardIt>::value_type in the range [first, first + count).

The operations in the parallel `destroy_n` algorithm invoked with an execution policy object of type `sequenced_policy` execute in sequential order in the calling thread.

**Note** Complexity: Performs exactly count operations, if count > 0, no assignments otherwise.

**Template Parameters**
- `ExPolicy`: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- `FwdIter`: The type of the source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.
- `Size`: The type of the argument specifying the number of elements to apply this algorithm to.

**Parameters**
- `policy`: The execution policy to use for the scheduling of the iterations.
- `first`: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- `count`: Refers to the number of elements starting at `first` the algorithm will be applied to.

The operations in the parallel `destroy_n` algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

**Return** The `destroy_n` algorithm returns a `hpx::future<FwdIter>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `FwdIter` otherwise. The `destroy_n` algorithm returns the iterator to the element in the source range, one past the last element constructed.

```cpp
#include <hpx/parallel/algorithms/adjacent_difference.hpp>

namespace hpx

namespace parallel

namespace v1

**Functions**

```cpp
template <typename ExPolicy, typename FwdIter1, typename FwdIter2>
```
Assigns each value in the range given by result its corresponding element in the range [first, last] and the one preceding it except *result, which is assigned *first

The difference operations in the parallel adjacent_difference invoked with an execution policy object of type sequenced_policy execute in sequential order in the calling thread.

**Note** Complexity: Exactly (last - first) - 1 application of the binary operator and (last - first) assignments.

**Template Parameters**
- ExPolicy: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- FwdIter1: The type of the source iterators used for the input range (deduced). This iterator type must meet the requirements of an forward iterator.
- FwdIter2: The type of the source iterators used for the output range (deduced). This iterator type must meet the requirements of an forward iterator.

**Parameters**
- policy: The execution policy to use for the scheduling of the iterations.
- first: Refers to the beginning of the sequence of elements of the range the algorithm will be applied to.
- last: Refers to the end of the sequence of elements of the range the algorithm will be applied to.
- dest: Refers to the beginning of the sequence of elements the results will be assigned to.

The difference operations in the parallel adjacent_difference invoked with an execution policy object of type parallel_policy or parallel_task_policy are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

This overload of adjacent_find is available if the user decides to provide their algorithm their own binary predicate op.

**Return**
The adjacent_difference algorithm returns a `hpx::future<FwdIter2>` if the execution policy is of type sequenced_task_policy or parallel_task_policy and returns FwdIter2 otherwise. The adjacent_find algorithm returns an iterator to the last element in the output range.

**template** `<typename ExPolicy, typename FwdIter1, typename FwdIter2, typename Op>
std::enable_if<execution::is_execution_policy<ExPolicy>::value, typename util::detail::algorithm_result<ExPolicy, FwdIter2>::type>::type
adjacent_difference(ExPolicy&& policy, FwdIter1 first, FwdIter1 last, FwdIter2 dest, Op&& op)"
The difference operations in the parallel `adjacent_difference` invoked with an execution policy object of type `sequenced_policy` execute in sequential order in the calling thread.

**Note** Complexity: Exactly (last - first) - 1 application of the binary operator and (last - first) assignments.

**Template Parameters**
- `ExPolicy`: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- `FwdIter1`: The type of the source iterators used for the input range (deduced). This iterator type must meet the requirements of a forward iterator.
- `FwdIter2`: The type of the source iterators used for the output range (deduced). This iterator type must meet the requirements of an forward iterator.
- `Op`: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of `adjacent_difference` requires `Op` to meet the requirements of `Copy-Constructible`.

**Parameters**
- `policy`: The execution policy to use for the scheduling of the iterations.
- `first`: Refers to the beginning of the sequence of elements of the range the algorithm will be applied to.
- `last`: Refers to the end of the sequence of elements of the range the algorithm will be applied to.
- `dest`: Refers to the beginning of the sequence of elements the results will be assigned to.
- `op`: The binary operator which returns the difference of elements. The signature should be equivalent to the following:

```cpp
bool op(const Type1 &a, const Type1 &b);
```

The signature does not need to have const &, but the function must not modify the objects passed to it. The types `Type1` must be such that objects of type `FwdIter1` can be dereferenced and then implicitly converted to the dereferenced type of `dest`.

The difference operations in the parallel `adjacent_difference` invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

**Return** The `adjacent_difference` algorithm returns a `hpx::future<FwdIter2>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `FwdIter2` otherwise. The `adjacent_find` algorithm returns an iterator to the last element in the output range.

```cpp
#include <hpx/parallel/algorithms/transform_exclusive_scan.hpp>

namespace hpx

namespace parallel

namespace v1

Functions

```
util::detail::algorithm_result<ExPolicy, FwdIter2>::type transform_exclusive_scan (ExPolicy &&policy, FwdIter1 first, FwdIter1 last, FwdIter2 dest, T init, Op &&op, Conv &&conv)

Assigns through each iterator \(i\) in \([\text{result}, \text{result} + (\text{last} - \text{first}))\) the value of GENERALIZED_NONCOMMUTATIVE_SUM(binary_op, init, conv(*first), \ldots, conv(*\text{(first} + (i - \text{result} - 1)))\).

The reduce operations in the parallel transform_exclusive_scan algorithm invoked with an execution policy object of type sequenced_policy execute in sequential order in the calling thread.

**Note** Complexity: \(O(\text{last} - \text{first})\) applications of the predicates \(op\) and \(conv\).

**Template Parameters**
- \(\text{ExPolicy}\): The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
- \(\text{FwdIter1}\): The type of the source iterators used (deduced). This iterator type must meet the requirements of an forward iterator.
- \(\text{FwdIter2}\): The type of the iterator representing the destination range (deduced). This iterator type must meet the requirements of an forward iterator.
- \(\text{Conv}\): The type of the unary function object used for the conversion operation.
- \(\text{T}\): The type of the value to be used as initial (and intermediate) values (deduced).
- \(\text{Op}\): The type of the binary function object used for the reduction operation.

**Parameters**
- \(\text{policy}\): The execution policy to use for the scheduling of the iterations.
- \(\text{first}\): Refers to the beginning of the sequence of elements the algorithm will be applied to.
- \(\text{last}\): Refers to the end of the sequence of elements the algorithm will be applied to.
- \(\text{dest}\): Refers to the beginning of the destination range.
- \(\text{conv}\): Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by \([\text{first}, \text{last})\). This is a unary predicate. The signature of this predicate should be equivalent to:

\[
R \ \text{fun}(\text{const Type } &a);
\]

The signature does not need to have \text{const}\&, but the function must not modify the objects passed to it. The type \text{Type} must be such that an object of type \text{FwdIter1} can be dereferenced and then implicitly converted to \text{Type}. The type \(R\) must be such that an object of this type can be implicitly converted to \(T\).
- \(\text{init}\): The initial value for the generalized sum.
- \(\text{op}\): Specifies the function (or function object) which will be invoked for each of the values of the input sequence. This is a binary predicate. The signature of this predicate should be equivalent to:
The signature does not need to have const&, but the function must not modify the objects passed to it. The types Type1 and Ret must be such that an object of a type as given by the input sequence can be implicitly converted to any of those types.

The reduce operations in the parallel transform_exclusive_scan algorithm invoked with an execution policy object of type parallel_policy or parallel_task_policy are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Neither conv nor op shall invalidate iterators or subranges, or modify elements in the ranges [first,last) or [result,result + (last - first)).

Return The copy_n algorithm returns a hpx::future<FwdIter2> if the execution policy is of type sequenced_task_policy or parallel_task_policy and returns FwdIter2 otherwise. The transform_exclusive_scan algorithm returns the output iterator to the element in the destination range, one past the last element copied.

Note GENERALIZED_NONCOMMUTATIVE_SUM(op, a1, ..., aN) is defined as:
- a1 when N is 1
- op(GENERALIZED_NONCOMMUTATIVE_SUM(op, a1, ..., aK), GENERALIZED_NONCOMMUTATIVE_SUM(op, aM, ..., aN)) where 1 < K+1 = M <= N.

The behavior of transform_exclusive_scan may be non-deterministic for a non-associative predicate.

```cpp
Ret fun(const Type1 &a, const Type1 &b);
```

The reduce operations in the parallel transform_exclusive_scan algorithm invoked with an execution policy object of type parallel_policy or parallel_task_policy are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

Neither conv nor op shall invalidate iterators or subranges, or modify elements in the ranges [first,last) or [result,result + (last - first)).

Return The copy_n algorithm returns a hpx::future<FwdIter2> if the execution policy is of type sequenced_task_policy or parallel_task_policy and returns FwdIter2 otherwise. The transform_exclusive_scan algorithm returns the output iterator to the element in the destination range, one past the last element copied.

Note GENERALIZED_NONCOMMUTATIVE_SUM(op, a1, ..., aN) is defined as:
- a1 when N is 1
- op(GENERALIZED_NONCOMMUTATIVE_SUM(op, a1, ..., aK), GENERALIZED_NONCOMMUTATIVE_SUM(op, aM, ..., aN)) where 1 < K+1 = M <= N.

The behavior of transform_exclusive_scan may be non-deterministic for a non-associative predicate.

```cpp
#include <hpx/parallel/algorithms/transform_reduce.hpp>

namespace hpx
{
  namespace parallel
  {
    namespace v1
    {
      Functions
      
      template <typename ExPolicy, typename FwdIter, typename T, typename Reduce, typename Convert>
      util::detail::algorithm_result<ExPolicy, T>::type transform_reduce (ExPolicy &&policy,
       FwdIter first, FwdIter last, T init, Reduce &&red_op, Convert &&conv_op)

      Returns GENERALIZED_SUM(red_op, init, conv_op(*first), ..., conv_op(*first + (last - first) - 1))).

      The reduce operations in the parallel transform_reduce algorithm invoked with an execution policy object of type sequenced_policy execute in sequential order in the calling thread.
      Note Complexity: O(last - first) applications of the predicates red_op and conv_op.
      
      Template Parameters
      • ExPolicy: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the assignments.
      • FwdIter: The type of the source iterators used (deduced). This iterator type must meet the requirements of an input iterator.
      • F: The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of copy_if requires F to meet the requirements of CopyConstructible.
      • T: The type of the value to be used as initial (and intermediate) values (deduced).
      ```
• **Reduce:** The type of the binary function object used for the reduction operation.
• **Convert:** The type of the unary function object used to transform the elements of the input sequence before invoking the reduce function.

**Parameters**

• **policy:** The execution policy to use for the scheduling of the iterations.
• **first:** Refers to the beginning of the sequence of elements the algorithm will be applied to.
• **last:** Refers to the end of the sequence of elements the algorithm will be applied to.
• **conv_op:** Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by [first, last). This is a unary predicate. The signature of this predicate should be equivalent to:

```cpp
R fun(const Type &a);
```

The signature does not need to have const&, but the function must not modify the objects passed to it. The type `Type` must be such that an object of type `FwdIter` can be dereferenced and then implicitly converted to `Type`. The type `R` must be such that an object of this type can be implicitly converted to `T`.

• **init:** The initial value for the generalized sum.
• **red_op:** Specifies the function (or function object) which will be invoked for each of the values returned from the invocation of `conv_op`. This is a binary predicate. The signature of this predicate should be equivalent to:

```cpp
Ret fun(const Type1 &a, const Type2 &b);
```

The signature does not need to have const&, but the function must not modify the objects passed to it. The types `Type1`, `Type2`, and `Ret` must be such that an object of a type as returned from `conv_op` can be implicitly converted to any of those types.

The reduce operations in the parallel `transform_reduce` algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

The difference between `transform_reduce` and `accumulate` is that the behavior of `transform_reduce` may be non-deterministic for non-associative or non-commutative binary predicate.

**Return** The `transform_reduce` algorithm returns a `hpx::future<T>` if the execution policy is of type `parallel_task_policy` and returns `T` otherwise. The `transform_reduce` algorithm returns the result of the generalized sum over the values returned from `conv_op` when applied to the elements given by the input range [first, last).

**Note** `GENERALIZED_SUM(op, a1, ..., aN)` is defined as follows:

- a1 when N is 1
- op(GENERALIZED_SUM(op, b1, ..., bK), GENERALIZED_SUM(op, bM, ..., bN)), where:
  - b1, ..., bN may be any permutation of a1, ..., aN and
  - 1 < K+1 = M <= N.

```cpp
#include <hpx/parallel/algorithms/sort.hpp>

namespace hpx
{

namespace parallel
{

namespace v1
{

```
Functions

```cpp
template <typename ExPolicy, typename RandomIt, typename Proj = util::projection_identity, typename Compare = detail::less>
util::detail::algorithm_result<ExPolicy, RandomIt>::type
sort(ExPolicy && policy, RandomIt first, RandomIt last, Compare && comp = Compare(), Proj && proj = Proj())
```

Sorts the elements in the range `[first, last)` in ascending order. The order of equal elements is not guaranteed to be preserved. The function uses the given comparison function object `comp` (defaults to using `operator<()`).

A sequence is sorted with respect to a comparator `comp` and a projection `proj` if for every iterator `i` pointing to the sequence and every non-negative integer `n` such that `i + n` is a valid iterator pointing to an element of the sequence, and `INVOKE(comp, INVOKE(proj, *(i + n)), INVOKE(proj, *i)) == false.`

**Note** Complexity: $O(N\log(N))$, where $N = \text{std::distance}(\text{first}, \text{last})$ comparisons.

`comp` has to induce a strict weak ordering on the values.

**Template Parameters**
- `ExPolicy`: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it applies user-provided function objects.
- `Iter`: The type of the source iterators used (deduced). This iterator type must meet the requirements of a random access iterator.
- `Comp`: The type of the function/function object to use (deduced).
- `Proj`: The type of an optional projection function. This defaults to `util::projection_identity`

**Parameters**
- `policy`: The execution policy to use for the scheduling of the iterations.
- `first`: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- `last`: Refers to the end of the sequence of elements the algorithm will be applied to.
- `comp`: `comp` is a callable object. The return value of the `INVOKE` operation applied to an object of type `Comp`, when contextually converted to bool, yields true if the first argument of the call is less than the second, and false otherwise. It is assumed that `comp` will not apply any non-constant function through the dereferenced iterator.
- `proj`: Specifies the function (or function object) which will be invoked for each pair of elements as a projection operation before the actual predicate `comp` is invoked.

The application of function objects in parallel algorithm invoked with an execution policy object of type `sequenced_policy` execute in sequential order in the calling thread.

The application of function objects in parallel algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

**Return** The `sort` algorithm returns a `hpx::future<RandomIt>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `RandomIt` otherwise. The algorithm returns an iterator pointing to the first element after the last element in the input sequence.

```cpp
#include <hpx/parallel/algorithms/count.hpp>

namespace hpx

namespace parallel

namespace v1
```
Functions

```cpp
template <typename ExPolicy, typename FwdIterB, typename FwdIterE, typename T, typename Proj = util::projection_identity>
util::detail::algorithm_result<ExPolicy, typename std::iterator_traits<FwdIterB>::difference_type>::type
count(
  ExPolicy &&policy,
  FwdIterB first,
  FwdIterE last,
  T const &value,
  Proj &&proj = Proj())
```

Returns the number of elements in the range [first, last) satisfying a specific criteria. This version counts the elements that are equal to the given `value`.

The comparisons in the parallel `count` algorithm invoked with an execution policy object of type `sequenced_policy` execute in sequential order in the calling thread.

**Note** Complexity: Performs exactly `last - first` comparisons.

**Template Parameters**

- `ExPolicy`: The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the comparisons.
- `FwdIterB`: The type of the source begin iterator used (deduced). This iterator type must meet the requirements of an forward iterator.
- `FwdIterE`: The type of the source end iterator used (deduced). This iterator type must meet the requirements of an forward iterator.
- `T`: The type of the value to search for (deduced).
- `Proj`: The type of an optional projection function. This defaults to `util::projection_identity`.

**Parameters**

- `policy`: The execution policy to use for the scheduling of the iterations.
- `first`: Refers to the beginning of the sequence of elements the algorithm will be applied to.
- `last`: Refers to the end of the sequence of elements the algorithm will be applied to.
- `value`: The value to search for.
- `proj`: Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate is invoked.

**Note** The comparisons in the parallel `count` algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

**Return** The `count` algorithm returns a `hpx::future<difference_type>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `difference_type` otherwise (where `difference_type` is defined by `std::iterator_traits<FwdIterB>::difference_type`). The `count` algorithm returns the number of elements satisfying the given criteria.
Returns the number of elements in the range [first, last) satisfying a specific criteria. This version counts elements for which predicate \( f \) returns true.

**Note** Complexity: Performs exactly \( last - first \) applications of the predicate.

**Note** The assignments in the parallel `count_if` algorithm invoked with an execution policy object of type `sequenced_policy` execute in sequential order in the calling thread.

**Note** The assignments in the parallel `count_if` algorithm invoked with an execution policy object of type `parallel_policy` or `parallel_task_policy` are permitted to execute in an unordered fashion in unspecified threads, and indeterminately sequenced within each thread.

**Return** The `count_if` algorithm returns `hpx::future<difference_type>` if the execution policy is of type `sequenced_task_policy` or `parallel_task_policy` and returns `difference_type` otherwise (where `difference_type` is defined by `std::iterator_traits<FwdIterB>::difference_type`). The `count` algorithm returns the number of elements satisfying the given criteria.

**Template Parameters**
- \( \text{ExPolicy} \): The type of the execution policy to use (deduced). It describes the manner in which the execution of the algorithm may be parallelized and the manner in which it executes the comparisons.
- \( \text{FwdIterB} \): The type of the source begin iterator used (deduced). This iterator type must meet the requirements of an forward iterator.
- \( \text{FwdIterE} \): The type of the source end iterator used (deduced). This iterator type must meet the requirements of an forward iterator.
- \( \text{F} \): The type of the function/function object to use (deduced). Unlike its sequential form, the parallel overload of `count_if` requires \( \text{F} \) to meet the requirements of `CopyConstructible`.
- \( \text{Proj} \): The type of an optional projection function. This defaults to `util::projection_identity`.

**Parameters**
- \( \text{policy} \): The execution policy to use for the scheduling of the iterations.
- \( \text{first} \): Refers to the beginning of the sequence of elements the algorithm will be applied to.
- \( \text{last} \): Refers to the end of the sequence of elements the algorithm will be applied to.
- \( \text{f} \): Specifies the function (or function object) which will be invoked for each of the elements in the sequence specified by \([\text{first}, \text{last})\). This is an unary predicate which returns \( true \) for the required elements. The signature of this predicate should be equivalent to:

```cpp
bool pred(const Type &a);
```

The signature does not need to have `const&`, but the function must not modify the objects passed to it. The type `Type` must be such that an object of type `FwdIterB` can be dereferenced and then implicitly converted to `Type`.

- \( \text{proj} \): Specifies the function (or function object) which will be invoked for each of the elements as a projection operation before the actual predicate \( is \) invoked.
#include <hpx/parallel/traits/projected.hpp>

```cpp
template <typename Iterator>
template<>
struct projected_iterator<Iterator, typename std::enable_if<is_segmented_iterator<Iterator>::value>::type>

Public Types

typedef segmented_iterator_traits<Iterator>::local_iterator local_iterator
typedef segmented_local_iterator_traits<local_iterator>::local_raw_iterator type
```

```cpp
template <typename Iterator>
template<>
struct projected_iterator<Iterator, typename hpx::util::always_void<type>::type>

Public Types

typedef hpx::util::decay<type>::proxy_type type
```

```cpp
namespace hpx

namespace parallel

namespace traits

```cpp
template <typename Proj, typename Iter>
struct projected

Public Types

typedef hpx::util::decay<Proj>::type projector_type
typedef hpx::traits::projected_iterator<Iter>::type iterator_type
```

```cpp
namespace traits

```cpp
template <typename T, typename Enable = void>
struct projected_iterator

Public Types

typedef hpx::util::decay<T>::type type
```

```cpp
template <typename Iterator>
template<>
struct projected_iterator<Iterator, typename hpx::util::always_void<type>::type>

Public Types

typedef hpx::util::decay<type>::proxy_type type
```

```cpp
template <typename Iterator>
template<>
struct projected_iterator<Iterator, typename std::enable_if<is_segmented_iterator<Iterator>::value>::type>
```
Public Types

typedef segmented_iterator_traits<Iterator>::local_iterator local_iterator

typedef segmented_local_iterator_traits<local_iterator>::local_raw_iterator type

#include <hpx/parallel/traits/projected_range.hpp>

template <typename Proj, typename Rng>
template<>
struct projected_range<Proj, Rng, typename std::enable_if<hpx::traits::is_range<Rng>::value>::type>

Public Types

typedef hpx::util::decay<Proj>::type projector_type

typedef hpx::traits::range_iterator<Rng>::type iterator_type

namespace hpx

namespace parallel

namespace traits

#include <hpx/parallel/datapar/iterator_helpers.hpp>

#include <hpx/parallel/datapar/loop.hpp>

#include <hpx/parallel/datapar/zip_iterator.hpp>

#include <hpx/parallel/datapar/transform_loop.hpp>

filesystem

#include <hpx/filesystem.hpp>

This file provides a compatibility layer using Boost.Filesystem for the C++17 filesystem library. It is not intended to be a complete compatibility layer. It only contains functions required by the HPX codebase. It also provides some functions only available in Boost.Filesystem when using C++17 filesystem.

namespace hpx

namespace filesystem
Functions

path initial_path()
path basename(path const &p)
path canonical(path const &p, path const &base)
path canonical(path const &p, path const &base, error_code &ec)

threadmanager
#include <compatibility/hpx/runtime/threads/threadmanager.hpp>
#include <hpx/threadmanager.hpp>

namespace hpx

namespace threads

class threadmanager
#include <threadmanager.hpp> The thread-manager class is the central instance of management for all (non-depleted) threads

Public Types

typedef threads::policies::callback_notifier notification_policy_type

typedef std::unique_ptr<thread_pool_base> pool_type

typedef threads::policies::scheduler_base scheduler_type

typedef std::vector<pool_type> pool_vector

Public Functions

threadmanager(notification_policy_type &notifier, detail::network_background_callback_type

~threadmanager()
void init()
void create_pools()
void print_pools(std::ostream&)

FIXME move to private and add hpx:printpools cmd line option.

thread_pool_base &default_pool() const
scheduler_type &default_scheduler() const

thread_pool_base &get_pool(std::string const &pool_name) const
thread_pool_base &get_pool(pool_id_type const &pool_id) const
thread_pool_base &get_pool(std::size_t thread_index) const
void register_work(thread_init_data &data, thread_state_enum initial_state = pending, error_code &ec = throws)

The function register_work adds a new work item to the thread manager. It doesn’t immediately create a new thread, it just adds the task parameters (function, initial state and description) to the internal management data structures. The thread itself will be created when the number of existing threads drops below the number of threads specified by the constructors max_count parameter.

Parameters

- **func**: [in] The function or function object to execute as the thread’s function. This must have a signature as defined by thread_function_type.
- **description**: [in] The value of this parameter allows to specify a description of the thread to create. This information is used for logging purposes mainly, but might be useful for debugging as well. This parameter is optional and defaults to an empty string.
- **initial_state**: [in] The value of this parameter defines the initial state of the newly created thread. This must be one of the values as defined by the thread_state enumeration (thread_state::pending, or thread_state::suspended, any other value will throw a hpx::bad_parameter exception).

void register_thread(thread_init_data &data, thread_id_type &id, thread_state_enum initial_state = pending, bool run_now = true, error_code &ec = throws)

The function register_thread adds a new work item to the thread manager. It creates a new thread, adds it to the internal management data structures, and schedules the new thread, if appropriate.

Parameters

- **func**: [in] The function or function object to execute as the thread’s function. This must have a signature as defined by thread_function_type.
- **id**: [out] This parameter will hold the id of the created thread. This id is guaranteed to be validly initialized before the thread function is executed.
- **description**: [in] The value of this parameter allows to specify a description of the thread to create. This information is used for logging purposes mainly, but might be useful for debugging as well. This parameter is optional and defaults to an empty string.
- **initial_state**: [in] The value of this parameter defines the initial state of the newly created thread. This must be one of the values as defined by the thread_state enumeration (thread_state::pending, or thread_state::suspended, any other value will throw a hpx::bad_parameter exception).
- **run_now**: [in] If this parameter is true and the initial state is given as thread_state::pending the thread will be run immediately, otherwise it will be scheduled to run later (either this function is called for another thread using true for the parameter run_now or the function threadmanager::do_some_work is called). This parameter is optional and defaults to true.

bool run()

Run the thread manager’s work queue. This function instantiates the specified number of OS threads in each pool. All OS threads are started to execute the function tfunc.

Return The function returns true if the thread manager has been started successfully, otherwise it returns false.

void stop(bool blocking = true)

Forcefully stop the thread-manager.

Parameters

- **blocking:*

void suspend()

void resume()
Return whether the thread manager is still running. This returns the “minimal state”, i.e. the state of the least advanced thread pool.

\[
\text{std::int64_t get_thread_count(thread_state_enum state = unknown, thread_priority priority = thread_priority_default, std::size_t num_thread = std::size_t(-1), bool reset = false)}
\]

return the number of HPX-threads with the given state

\textbf{Note} This function lock the internal OS lock in the thread manager.

\[
\text{std::int64_t get_background_thread_count()}
\]

bool enumerate_threads (util::function_nonser<bool thread_id_type > const &f, thread_state_enum state = unknown const)

void abort_all_suspended_threads ()

bool cleanup_terminated (bool delete_all)

\[
\text{std::size_t get_os_thread_count() const}
\]

Return the number of OS threads running in this thread-manager. This function will return correct results only if the thread-manager is running.

\[
\text{std::thread &get_os_thread_handle(std::size_t num_thread) const}
\]

void report_error (std::size_t num_thread, std::exception_ptr const &e)

API functions forwarding to notification policy.

This notifies the thread manager that the passed exception has been raised. The exception will be routed through the notifier and the scheduler (which will result in it being passed to the runtime object, which in turn will report it to the console, etc.).

mask_type get_used_processing_units () const

Returns the mask identifying all processing units used by this thread manager.

hwloc_bitmap_ptr get_pool_numa_bitmap (const std::string &pool_name) const

void set_scheduler_mode (threads::policies::scheduler_mode mode)

void add_scheduler_mode (threads::policies::scheduler_mode mode)

void add_remove_scheduler_mode (threads::policies::scheduler_mode to_add_mode, threads::policies::scheduler_mode to_remove_mode)

void remove_scheduler_mode (threads::policies::scheduler_mode mode)

void reset_thread_distribution ()

void init_tss (std::size_t global_thread_num)

void deinit_tss ()

\[
\text{std::size_t shrink_pool(std::string const &pool_name)}
\]

\[
\text{std::size_t expand_pool(std::string const &pool_name)}
\]

\[
\text{std::int64_t get_queue_length(bool reset)}
\]
std::int64_t get_cumulative_duration(bool reset)
std::int64_t get_thread_count_unknown(bool reset)
std::int64_t get_thread_count_active(bool reset)
std::int64_t get_thread_count_pending(bool reset)
std::int64_t get_thread_count_suspended(bool reset)
std::int64_t get_thread_count_terminated(bool reset)
std::int64_t get_thread_count_staged(bool reset)

Private Types

typedef std::mutex mutex_type

Private Members

mutex_type mtx_
std::size_t num_threads_
std::vector<pool_id_type> threads_lookup_
pool_vector pools_
notification_policy_type &notifier_
detail::network_background_callback_type network_background_callback_

static_reinit

#include <compatibility/hpx/util/reinitializable_static.hpp>
#include <compatibility/hpx/util/static_reinit.hpp>
#include <hpx/static_reinit/reinitializable_static.hpp>

Defines

HPX_EXPORT_REINITIALIZABLE_STATIC
namespace hpx

namespace util

Variables

struct HPX_EXPORT_REINITIALIZABLE_STATIC reinitializable_static

template <typename T, typename Tag, std::size_t N>
struct reinitializable_static
Public Types

typedef T value_type

typedef T & reference

typedef T const & const_reference

Public Functions

HPX_NON_COPYABLE (reinitializable_static)

reinitializable_static()

template <typename U>
reinitializable_static(U const & val)

operator reference()

operator const_reference() const

reference get (std::size_t item = 0)

const_reference get (std::size_t item = 0) const

Private Types

typedef std::add_pointer<value_type>::type pointer

typedef std::aligned_storage<sizeof (value_type),
std::alignment_of<value_type>::value>::type storage_type

Private Static Functions

static void default_construct ()

template <typename U>
static void value_construct (U const & v)

static void destruct ()

static void default_constructor ()

template <typename U>
static void value_constructor (U const * pv)

static pointer get_address (std::size_t item)

Private Static Attributes

reinitializable_static<T, Tag, N>::storage_type data_

std::once_flag constructed_

#include <hpx/static_reinit/static_reinit.hpp>

namespace hpx

namespace util
Functions

void reinit_register (util::function_nonser<void>)
  > const &constructutil::function_nonser<void> const &destruct

void reinit_construct ()

void reinit_destruct ()

thread_support

#include <compatibility/hpx/util/thread_specific_ptr.hpp>
#include <compatibility/hpx/util/set_thread_name.hpp>
#include <compatibility/hpx/util/atomic_count.hpp>
#include <compatibility/hpx/util/unlock_guard.hpp>
#include <compatibility/hpx/util/assert_owns_lock.hpp>
#include <hpx/thread_support/thread_specific_ptr.hpp>

Defines

HPX_EXPORT_THREAD_SPECIFIC_PTR

namespace hpx

namespace util

  template <typename T, typename Tag>
  struct thread_specific_ptr

    Public Types

    typedef boost::thread_specific_ptr<T>::element_type element_type

    Public Functions

    T *get () const
    T *operator-> () const
    T &operator* () const
    void reset (T *new_value = nullptr)

    Private Static Attributes

    boost::thread_specific_ptr<T> ptr_
```cpp
#include <hpx/thread_support/set_thread_name.hpp>

namespace hpx
{
    namespace util
    {
        Functions
        {
            void set_thread_name(char const *threadName);
        }
    }
}

#include <hpx/thread_support/atomic_count.hpp>

namespace hpx
{
    namespace util
    {
        class atomic_count
        {
            Public Functions
            {
                HPX_NON_COPYABLE(atomic_count)
                atomic_count(long value);
                atomic_count &operator=(long value);
                long operator++();
                long operator--();
                atomic_count &operator+=(long n);
                atomic_count &operator-=(long n);
                operator long() const;
            }
            Private Members
            {
                std::atomic<long> value_
            }
        }
    }
}

#include <hpx/thread_support/unlock_guard.hpp>

namespace hpx
{
    namespace util
    {
        template<typename Mutex>
        class unlock_guard
        {
            Public Types
            {
                template<>
                using mutex_type = Mutex;
            }
        }
    }
}
```
Public Functions

HPX_NON_COPYABLE (unlock_guard)

unlock_guard (Mutex &m)
~unlock_guard ()

Private Members

Mutex &m_

#include <hpx/thread_support/assert_owns_lock.hpp>

Defines

HPX_ASSERT_OWNS_LOCK (l)

compute_cuda

#include <hpx/compute/cuda.hpp>
#include <hpx/compute/cuda/transfer.hpp>
#include <hpx/compute/cuda/default_executor.hpp>
#include <hpx/compute/cuda/target.hpp>
#include <hpx/compute/cuda/default_executor_parameters.hpp>
#include <hpx/compute/cuda/concurrent_executor.hpp>
#include <hpx/compute/cuda/target_distribution_policy.hpp>
#include <hpx/compute/cuda/target_ptr.hpp>
#include <hpx/compute/cuda/get_targets.hpp>
#include <hpx/compute/cuda/allocator.hpp>
#include <hpx/compute/cuda/value_proxy.hpp>
#include <hpx/compute/cuda/concurrent_executor_parameters.hpp>
#include <hpx/compute/cuda/traits/access_target.hpp>
#include <hpx/compute/cuda/serialization/value_proxy.hpp>

2.10 Contributing to HPX

HPX development happens on Github. The following sections are a collection of useful information related to HPX development.

2.10.1 Release procedure for HPX

Below is a step by step procedure for making an HPX release. We aim to produce two releases per year: one in March-April, and one in September-October.
This is a living document and may not be totally current or accurate. It is an attempt to capture current practices in making an HPX release. Please update it as appropriate.

One way to use this procedure is to print a copy and check off the lines as they are completed to avoid confusion.

1. Notify developers that a release is imminent.

2. Make a list of examples and benchmarks that should not go into the release. Build all examples and benchmarks that will go in the release and make sure they build and run as expected.
   - Make sure all examples and benchmarks have example input files, and usage documentation, either in the form of comments or a readme.

3. Send the list of examples and benchmarks that will be included in the release to hpx-users@stellar.cct.lsu.edu and stellar@cct.lsu.edu, and ask for feedback. Update the list as necessary.

4. Write release notes in docs/sphinx/releases/whats_new_$VERSION.rst. Keep adding merged PRs and closed issues to this until just before the release is made. Use tools/generate_pr_issue_list.sh to generate the lists. Add the new release notes to the table of contents in docs/sphinx/releases.rst.

5. Build the docs, and proof-read them. Update any documentation that may have changed, and correct any typos. Pay special attention to:
   - $HPX_SOURCE/README.rst
     - Update grant information
   - docs/sphinx/releases/whats_new_$VERSION.rst
   - docs/sphinx/about_hpx/people.rst
     - Update collaborators
     - Update grant information

6. This step does not apply to patch releases. For both APEX and hpxMP:
   - Change the release branch to be the most current release tag available in the APEX/hpxMP git_external section in the main CMakeLists.txt. Please contact the maintainers of the respective packages to generate a new release to synchronize with the HPX release (APEX\textsuperscript{230}, hpxMP\textsuperscript{231}).

7. If there have been any commits to the release branch since the last release, create a tag from the old release branch before deleting the old release branch in the next step.

8. Unprotect the release branch in the github repository settings so that it can be deleted and recreated.

9. Delete the old release branch, and create a new one by branching a stable point from master. If you are creating a patch release, branch from the release tag for which you want to create a patch release.
   - git push origin --delete release
   - git branch -D release
   - git checkout [stable point in master]
   - git branch release
   - git push origin release
   - git branch --set-upstream-to=origin/release release

10. Protect the release branch again to disable deleting and force pushes.

11. Check out the release branch.

\textsuperscript{230} http://github.com/khuck/xpress-apex
\textsuperscript{231} https://github.com/STEllAR-GROUP/hpxMP
12. Make sure HPX_VERSION_MAJOR/MINOR/SUBMINOR in CMakeLists.txt contain the correct values. Change them if needed.

13. Remove the examples and benchmarks that will not go into the release from the release branch.

14. This step does not apply to patch releases. Remove features which have been deprecated for at least 2 releases. This involves removing build options which enable those features from the main CMakeLists.txt and also deleting all related code and tests from the main source tree.

The general deprecation policy involves a three-step process we have to go through in order to introduce a breaking change:

(a) First release cycle: add a build option that allows for explicitly disabling any old (now deprecated) code.
(b) Second release cycle: turn this build option OFF by default.
(c) Third release cycle: completely remove the old code.

The main CMakeLists.txt contains a comment indicating for which version the breaking change was introduced first.

15. Switch Buildbot over to test the release branch.

   • https://github.com/STEllAR-GROUP/hermione-buildbot/blob/rostam/master/master.cfg
   • branch field in c['change_source'] = GitPoller

16. Repeat the following steps until satisfied with the release.

(a) Change HPX_VERSION_TAG in CMakeLists.txt to -rcN, where N is the current iteration of this step. Start with -rc1.
(b) Tag and create a pre-release on GitHub using the script tools/roll_release.sh. The script requires that you have the STEllAR Group signing key.
(c) This step is not necessary for patch releases. Notify hpx-users@stellar.cct.lsu.edu and stellar@cct.lsu.edu of the availability of the release candidate. Ask users to test the candidate by checking out the release candidate tag.
(d) Allow at least a week for testing of the release candidate.
   • Use git merge when possible, and fall back to git cherry-pick when needed. For patch releases git cherry-pick is most likely your only choice if there have been significant unrelated changes on master since the previous release.
   • Go back to the first step when enough patches have been added.
   • If there are no more patches, continue to make the final release.

17. Update any occurrences of the latest stable release to refer to the version about to be released. For example, quickstart.rst contains instructions to check out the latest stable tag. Make sure that refers to the new version.

18. Add a new entry to the RPM changelog (cmake/packaging/rpm/Changelog.txt) with the new version number and a link to the corresponding changelog.

19. Change HPX_VERSION_TAG in CMakeLists.txt to an empty string.

20. Add the release date to the caption of the current “What’s New” section in the docs, and change the value of HPX_VERSION_DATE in CMakeLists.txt.

21. Tag and create a release on GitHub using the script tools/roll_release.sh. The script requires that you have the STEllAR Group signing key.
22. Update the websites (stellar-group.org\textsuperscript{232} and stellar.cct.lsu.edu\textsuperscript{233}) with the following:

- Download links on the downloads pages. Link to the release on GitHub.
- Documentation links on the docs page (link to generated documentation on GitHub Pages). Follow the style of previous releases.
- A new blog post announcing the release, which links to downloads and the “What’s New” section in the documentation (see previous releases for examples).

23. Merge release branch into master.

24. This step does not apply to patch releases. Bump version numbers on master.

(a) Create a new branch from master, and check that branch out (name it, for example, by the next version number).

(b) Bump the HPX version to the next release target. The following files contain version info:
   - CMakeLists.txt
   - Grep for old version number

(c) Create a new “What’s New” section for the docs of the next anticipated release. Set the date to “unreleased”. Make sure you add it to the table of contents in docs/sphinx/releases.rst.

(d) Modify the the release procedure if necessary.

(e) Merge new branch containing next version numbers to master; resolve conflicts if necessary.

25. Switch Buildbot back to test the main branch

   - https://github.com/STEllAR-GROUP/hermione-buildbot/blob/rostam/master/master.cfg
   - branch field in c['change_source'] = GitPoller


   - Update version number in CONTROL
   - Update tag and SHA512 to that of the new release

27. Announce the release on hpx-users@stellar.cct.lsu.edu, stellar@cct.lsu.edu, allcct@cct.lsu.edu, faculty@csc.lsu.edu, faculty@ece.lsu.edu, xpress@crest.iu.edu, the HPX Slack channel, the IRC channel, Sonia Sachs, our list of external collaborators, isocpp.org, reddit.com, HPC Wire, Inside HPC, Heise Online, and a CCT press release.


2.10.2 Testing HPX

To ensure correctness of HPX, we ship a large variety of unit and regression tests. The tests are driven by the CTest\textsuperscript{234} tool and are executed automatically by buildbot (see HPX Buildbot Website\textsuperscript{235}) on each commit to the HPX Github\textsuperscript{236} repository. In addition, it is encouraged to run the test suite manually to ensure proper operation on your target system. If a test fails for your platform, we highly recommend submitting an issue on our HPX Issues\textsuperscript{237} tracker with detailed information about the target system.

\textsuperscript{232} https://stellar-group.org
\textsuperscript{233} https://stellar.cct.lsu.edu
\textsuperscript{234} https://gitlab.kitware.com/cmake/community/wikis/doc/ctest/Testing-With-CTest
\textsuperscript{235} http://rostam.cct.lsu.edu/
\textsuperscript{236} https://github.com/STEllAR-GROUP/hpx/
\textsuperscript{237} https://github.com/STEllAR-GROUP/hpx/issues
Running tests manually

Running the tests manually is as easy as typing `make tests && make test`. This will build all tests and run them once the tests are built successfully. After the tests have been built, you can invoke separate tests with the help of the `ctest` command. You can list all available test targets using `make help | grep tests`. Please see the CTest Documentation\(^\text{238}\) for further details.

Issue tracker

If you stumble over a bug or missing feature in HPX, please submit an issue to our HPX Issues\(^\text{239}\) page. For more information on how to submit support requests or other means of getting in contact with the developers, please see the Support Website\(^\text{240}\) page.

Continuous testing

In addition to manual testing, we run automated tests on various platforms. You can see the status of the current master head by visiting the HPX Buildbot Website\(^\text{241}\). We also run tests on all pull requests using both CircleCI\(^\text{242}\) and a combination of CDash\(^\text{243}\) and pycicle\(^\text{244}\). You can see the dashboards here: CircleCI HPX dashboard\(^\text{245}\) and CDash HPX dashboard\(^\text{246}\).

2.10.3 Using docker for development

Although it can often be useful to set up a local development environment with system-provided or self-built dependencies, Docker\(^\text{247}\) provides a convenient alternative to quickly get all the dependencies needed to start development of HPX. Our testing setup on CircleCI\(^\text{248}\) uses a docker image to run all tests.

To get started you need to install Docker\(^\text{249}\) using whatever means is most convenient on your system. Once you have Docker\(^\text{250}\) installed, you can pull or directly run the docker image. The image is based on Debian and Clang, and can be found on Docker Hub\(^\text{251}\). To start a container using the HPX build environment, run:

```bash
docker run --interactive --tty stellargroup/build_env:ubuntu bash
```

You are now in an environment where all the HPX build and runtime dependencies are present. You can install additional packages according to your own needs. Please see the Docker Documentation\(^\text{252}\) for more information on using Docker\(^\text{253}\).

**Warning:** All changes made within the container are lost when the container is closed. If you want files to persist (e.g., the HPX source tree) after closing the container, you can bind directories from the host system into the container (see Docker Documentation (Bind mounts)\(^\text{254}\)).

---

239 https://github.com/STEllAR-GROUP/hpx/issues
240 https://stellar.cct.lsu.edu/support/
241 http://rostam.cct.lsu.edu/
242 https://circleci.com
243 https://www.kitware.com/cdash/project/about.html
244 https://github.com/biddisco/pycicle/
245 https://circleci.com/gh/STEllAR-GROUP/hpx
246 https://cdash.csrs.ch/index.php?project=HPX
247 https://www.docker.com
248 https://circleci.com
249 https://www.docker.com
250 https://www.docker.com
251 https://hub.docker.com/r/stellargroup/build_env/
252 https://docs.docker.com/
253 https://www.docker.com
254 https://docs.docker.com/storage/bind-mounts/
2.10.4 Documentation

This documentation is built using Sphinx\(^{255}\), and an automatically generated API reference using Doxygen\(^{256}\) and Breathe\(^{257}\).

We always welcome suggestions on how to improve our documentation, as well as pull requests with corrections and additions.

Building documentation

Please see the documentation prerequisites section for details on what you need in order to build the HPX documentation. Enable building of the documentation by setting HPX\_WITH\_DOCUMENTATION=ON during CMake\(^{258}\) configuration. To build the documentation, build the docs target using your build tool. The default output format is HTML documentation. You can choose alternative output formats (single-page HTML, PDF, and man) with the HPX\_WITH\_DOCUMENTATION\_OUTPUT\_FORMATS CMake option.

Note: If you add new source files to the Sphinx documentation, you have to run CMake again to have the files included in the build.

Style guide

The documentation is written using reStructuredText. These are the conventions used for formatting the documentation:

- Use, at most, 80 characters per line.
- Top-level headings use over- and underlines with =.
- Sub-headings use only underlines with characters in decreasing level of importance: =, – and ..
- Use sentence case in headings.
- Refer to common terminology using :term:`Component`.
- Indent content of directives (:ref:`directive::<ref>` by three spaces.
- For C++ code samples at the end of paragraphs, use :: and indent the code sample by 4 spaces.
  - For other languages (or if you don’t want a colon at the end of the paragraph), use .. code-block::<language> and indent by three spaces as with other directives.
- Use .. list-table:: to wrap tables with a lot of text in cells.

API documentation

The source code is documented using Doxygen. If you add new API documentation either to existing or new source files, make sure that you add the documented source files to the doxygen_dependencies variable in docs/CMakeLists.txt.

2.10.5 Module structure

This section explains the structure of an HPX module.

The tool \texttt{create\_library\_skeleton.py}\(^{259}\) can be used to generate a basic skeleton. To create a library skeleton, run the tool in the \texttt{libs} subdirectory with the module name as an argument:

\(^{255}\) http://www.sphinx-doc.org
\(^{256}\) https://www.doxygen.org
\(^{257}\) https://breathe.readthedocs.io/en/latest
\(^{258}\) https://www.cmake.org
\(^{259}\) https://github.com/STEllAR-GROUP/hpx/blob/master/libs/create\_library\_skeleton.py
This creates a skeleton with the necessary files for an HPX module. It will not create any actual source files. The structure of this skeleton is as follows:

- `<lib_name>/`
  - README.rst
  - CMakeLists.txt
  - cmake
  - docs/
    - index.rst
  - examples/
    - CMakeLists.txt
- include/
  - hpx/
    - `<lib_name>`
- src/
  - CMakeLists.txt
- tests/
  - CMakeLists.txt
  - unit/
    - CMakeLists.txt
  - regressions/
    - CMakeLists.txt
  - performance/
    - CMakeLists.txt

A README.rst should be always included which explains the basic purpose of the library and a link to the generated documentation.

A main CMakeLists.txt is created in the root directory of the module. By default it contains a call to `add_hpx_module` which takes care of most of the boilerplate required for a module. You only need to fill in the source and header files in most cases.

`add_hpx_module` requires a module name. Optional flags are:

- DEPRECATION_WARNINGS: Enables deprecation warnings for the module.

Optional single-value arguments are:

- COMPATIBILITY_HEADERS: Can be ON, OFF, or left out. Enables compatibility headers. Creates a variable which can be turned on or off by the user when set to ON or OFF. If left out the option is completely disabled.

- INSTALL_BINARIES: Install the resulting library.

Optional multi-value arguments are:

- SOURCES: List of source files.

2.10. Contributing to HPX
- **HEADERS**: List of header files.
- **COMPAT_HEADERS**: List of compatibility header files.
- **DEPENDENCIES**: Libraries that this module depends on, such as other modules.
- **CMAKE_SUBDIRS**: List of subdirectories to add to the module.

The `include` directory should contain only headers that other libraries need. For each of those headers, an automatic header test to check for self containment will be generated. Private headers should be placed under the `src` directory. This allows for clear separation. The `cmake` subdirectory may include additional CMake scripts needed to generate the respective build configurations.

Compatibility headers (forwarding headers for headers whose location is changed when creating a module, if moving them from the main library) should be placed in an `include_compatibility` directory. This directory is not created by default.

Documentation is placed in the `docs` folder. A empty skeleton for the index is created, which is picked up by the main build system and will be part of the generated documentation. Each header inside the `include` directory will automatically be processed by Doxygen and included into the documentation. If a header should be excluded from the API reference, a comment `// sphinx:undocumented` needs to be added.

Tests are placed in suitable subdirectories of `tests`.

When in doubt, consult existing modules for examples on how to structure the module.

## 2.11 Releases

### 2.11.1 HPX V1.4.1 (Feb 19, 2020)

**General changes**

This is a bugfix release. It contains the following changes:

- Fix compilation issues on Windows, macOS, FreeBSD, and with gcc 10
- Install missing `pdb` files on Windows
- Allow running tests using an installed version of `HPX`
- Skip MPI finalization if HPX has not initialized MPI
- Give a hard error when attempting to use IO counters on Windows
- Fix CMake configuration using older CMake versions with CUDA enabled
- Fix installation of `pkgconfig` files when build type is typed in unexpected case
- Add `jsrun` option to `hpxrun.py`
- Fix cases of scheduling hints not being taken into account

**Closed issues**

- Issue #4320\(^{261}\) - HPX 1.4.0 does not compile with gcc 10
- Issue #4336\(^{262}\) - Building HPX 1.4.0 with IO Counters breaks (Windows)
- Issue #4334\(^{263}\) - HPX Debug and RelWithDebinfo builds on Windows not installing `.pdb` files

---

\(^{260}\) [https://www.cmake.org](https://www.cmake.org)

\(^{261}\) [https://github.com/STEllAR-GROUP/hpx/issues/4320](https://github.com/STEllAR-GROUP/hpx/issues/4320)

\(^{262}\) [https://github.com/STEllAR-GROUP/hpx/issues/4336](https://github.com/STEllAR-GROUP/hpx/issues/4336)

\(^{263}\) [https://github.com/STEllAR-GROUP/hpx/issues/4334](https://github.com/STEllAR-GROUP/hpx/issues/4334)
• Issue #4322264 - Undefine VT1 and VT2 after boost includes
• Issue #4314265 - Compile error on 1.4.0
• Issue #4307266 - ld: error: duplicate symbol: freebsd_environ

Closed pull requests

• PR #4396267 - Relax build type matching when generating pkgconfig files
• PR #4389268 - Add jsrun in hpx run wrapper
• PR #4376269 - Attempt to fix some test build errors on Windows
• PR #4357270 - Adding missing #includes to fix gcc V10 linker problems
• PR #4353271 - Skip MPI_Finalize if MPI_Init is not called from HPX
• PR #4343272 - Give a hard error if IO counters are enabled on non-Linux systems
• PR #4337273 - Installing pdb files on Windows
• PR #4335274 - Adding capability to buildsystem to use an installed version of HPX
• PR #4315275 - Forcing exported symbols from composable_guard to be linked into core library
• PR #4310276 - Remove environment handling from exception.cpp
• PR #4306277 - Fix schedule hint not being taken from executor

2.11.2 HPX V1.4.0 (January 15, 2020)

General changes

• We have added the collectives all_to_all and all_reduce.
• We have added APIs for resiliency, which allows replication and replay for failed tasks. See the documentation for more details.
• Components can now be checkpointed.
• Performance improvements to schedulers and coroutines. A significant change is the addition of stackless coroutines. These are to be used for tasks that do not need to be suspended and can reduce overheads noticeably in applications with short tasks. A stackless coroutine can be created with the new stack size thread_stacksize_nostack.
• We have added an implementation of unique_any, which is a non-copyable version of any.
• The shared_priority_queue_scheduler has been improved. It now has lower overheads than the default scheduler in many situations. Unlike the default scheduler it fully supports NUMA scheduling hints.
Enable it with the command line option `--hpx:queuing=shared-priority`. This scheduler should still be considered experimental, but its use is encouraged in real applications to help us make it production ready.

- We have added the performance counters `background-receive-duration` and `background-receive-overhead` for inspecting the time and overhead spent on receiving parcels in the background.
- Compilation time has been further improved when `HPX_WITH_NETWORKING=OFF`.
- We no longer require compiled Boost dependencies in certain configurations. This requires at least Boost 1.70, compiling on x86 with GCC 9, clang (libc++) 9, or VS2019 in C++17 mode. The dependency on Boost.Filesystem can explicitly be turned on with `HPX_FILESYSTEM_WITH_BOOST_FILESYSTEM_COMPATIBILITY=ON` (it is off by default if the standard library supports `std::filesystem`). Boost.ProgramOptions has been copied into the HPX repository. We have a compatibility layer for users who must explicitly use Boost.ProgramOptions instead of the ProgramOptions provided by HPX. To remove the dependency `HPX_PROGRAM_OPTIONS_WITH_BOOST_PROGRAM_OPTIONS_COMPATIBILITY` must be explicitly set to `OFF`. This option will be removed in a future release. We have also removed several other header-only dependencies on Boost.
- It is now possible to use the process affinity mask set by tools like `numactl` and various batch environments with the command line option `--hpx:use-process-mask`. Enabling this option implies `--hpx:ignore-batch-env`.
- It is now possible to create standalone thread pools without starting the runtime. See the `standalone_thread_pool_executor.cpp` test in the execution module for an example.
- Tasks annotated with `hpx::util::annotated_function` now have their correct name when using APEX to generate OTF2 files.
- Cloning of APEX was defective in previous releases (it required manual intervention to check out the correct tag or branch). This has been fixed.
- The option `HPX_WITH_MORE_THAN_64_THREADS` is now ignored and will be removed in a future release. The value is instead derived directly from `HPX_WITH_MAX_CPU_COUNT` option.
- We have deprecated compiling in C++11 mode. The next release will require a C++14 capable compiler.
- We have deprecated support for the Vc library. This option will be replaced with SIMD support from the standard library in a future release.
- We have significantly refactored our CMake setup. This is intended to be a non-breaking change and will allow for using HPX through CMake targets in the future.
- We have continued modularizing the HPX library. In the process we have rearranged many header files into module-specific directories. All moved headers have compatibility headers which forward from the old location to the new location, together with a deprecation warning. The compatibility headers will eventually be removed.
- We now enforce formatting with `clang-format` on the majority of our source files.
- We have added SPDX license tags to all files.
- Many bugfixes.

**Breaking changes**

- The `HPX_WITH_THREAD_COMPATIBILITY` option and the associated compatibility layer has been removed.
- The `HPX_WITH_INCLUSIVE_SCAN_COMPATIBILITY` option and the associated compatibility layer has been removed.
- The `HPX_WITH_UNWRAPPED_COMPATIBILITY` option and the associated compatibility layer has been removed.
Closed issues

- Issue #4282 - Build Issues with Release on Windows
- Issue #4279 - Build Issues with CMake 3.14.4
- Issue #4273 - Clients of HPX 1.4.0-rc2 with APEX ar not linked to libhpx-apex
- Issue #4269 - Building HPX 1.4.0-rc2 with support for APEX fails
- Issue #4263 - Compilation fail on latest master
- Issue #4232 - Configure of HPX project using CMake FetchContent fails
- Issue #4223 - “Re-using the main() function as the main HPX entry point” doesn’t work
- Issue #4220 - HPX won’t compile - error building resource_partitioner
- Issue #4215 - HPX 1.4.0rc1 does not link on s390x
- Issue #4204 - Trouble compiling HPX with Intel compiler
- Issue #4199 - Refactor APEX to eliminate circular dependency
- Issue #4187 - HPX can’t build on OSX
- Issue #4185 - Simple debug output for development
- Issue #4182 - @HPX_CONF_PREFIX@ is the empty string
- Issue #4169 - HPX won’t build with APEX
- Issue #4163 - Add back HPX_LIBRARIES and HPX_INCLUDE_DIRS
- Issue #4161 - It should be possible to call find_package (HPX) multiple times
- Issue #4155 - get_self_id() for stackless threads returns invalid_thread_id
- Issue #4151 - build error with MPI code
- Issue #4150 - hpx won’t build on POWER9 with clang 8
- Issue #4148 - cacheline data delivers poor performance with C++17 compared to C++14
- Issue #4144 - target general in HPX_LIBRARIES does not exist

https://github.com/STEllAR-GROUP/hpx/issues/4282
https://github.com/STEllAR-GROUP/hpx/issues/4279
https://github.com/STEllAR-GROUP/hpx/issues/4273
https://github.com/STEllAR-GROUP/hpx/issues/4269
https://github.com/STEllAR-GROUP/hpx/issues/4263
https://github.com/STEllAR-GROUP/hpx/issues/4232
https://github.com/STEllAR-GROUP/hpx/issues/4223
https://github.com/STEllAR-GROUP/hpx/issues/4215
https://github.com/STEllAR-GROUP/hpx/issues/4204
https://github.com/STEllAR-GROUP/hpx/issues/4199
https://github.com/STEllAR-GROUP/hpx/issues/4187
https://github.com/STEllAR-GROUP/hpx/issues/4169
https://github.com/STEllAR-GROUP/hpx/issues/4182
https://github.com/STEllAR-GROUP/hpx/issues/4163
https://github.com/STEllAR-GROUP/hpx/issues/4161
https://github.com/STEllAR-GROUP/hpx/issues/4155
https://github.com/STEllAR-GROUP/hpx/issues/4150
https://github.com/STEllAR-GROUP/hpx/issues/4148
https://github.com/STEllAR-GROUP/hpx/issues/4144
- Issue #4134\(^{300}\) - CMake Error when -DHPX\_WITH\_HPXMP=ON
- Issue #4132\(^{301}\) - parallel fill leaves elements unfilled
- Issue #4123\(^{302}\) - PAPI performance counters are inaccessible
- Issue #4118\(^{303}\) - static\_chunk\_size is not obeyed in scan algorithms
- Issue #4115\(^{304}\) - dependency chaining error with APEX
- Issue #4107\(^{305}\) - Initializing runtime without entry point function and command line arguments
- Issue #4105\(^{306}\) - Bug in hpx:bind= numa-balanced
- Issue #4101\(^{307}\) - Bound tasks
- Issue #4100\(^{308}\) - Add SPDX identifier to all files
- Issue #4085\(^{309}\) - hpx\_topology library should depend on hwloc
- Issue #4067\(^{310}\) - HPX fails to build on macOS
- Issue #4056\(^{311}\) - Building without thread manager idle backoff fails
- Issue #4052\(^{312}\) - Enforce clang\_format style for modules
- Issue #4032\(^{313}\) - Simple hello world fails to launch correctly
- Issue #4030\(^{314}\) - Allow threads to skip context switching
- Issue #4029\(^{315}\) - Add support for mimalloc
- Issue #4005\(^{316}\) - Can’t link HPX when APEX enabled
- Issue #4002\(^{317}\) - Missing header for algorithm module
- Issue #3989\(^{318}\) - conversion from long to unsigned int requires a narrowing conversion on MSVC
- Issue #3958\(^{319}\) - /statistics/average@ perf counter can’t be created
- Issue #3953\(^{320}\) - CMake errors from HPX\_AddPseudoDependencies
- Issue #3941\(^{321}\) - CMake error for APEX install target
- Issue #3940\(^{322}\) - Convert pseudo-doxygen function documentation into actual doxygen documentation

\(^{300}\) https://github.com/STEllAR-GROUP/hpx/issues/4134
\(^{301}\) https://github.com/STEllAR-GROUP/hpx/issues/4132
\(^{302}\) https://github.com/STEllAR-GROUP/hpx/issues/4123
\(^{303}\) https://github.com/STEllAR-GROUP/hpx/issues/4118
\(^{304}\) https://github.com/STEllAR-GROUP/hpx/issues/4115
\(^{305}\) https://github.com/STEllAR-GROUP/hpx/issues/4107
\(^{306}\) https://github.com/STEllAR-GROUP/hpx/issues/4105
\(^{307}\) https://github.com/STEllAR-GROUP/hpx/issues/4101
\(^{308}\) https://github.com/STEllAR-GROUP/hpx/issues/4100
\(^{309}\) https://github.com/STEllAR-GROUP/hpx/issues/4085
\(^{310}\) https://github.com/STEllAR-GROUP/hpx/issues/4067
\(^{311}\) https://github.com/STEllAR-GROUP/hpx/issues/4056
\(^{312}\) https://github.com/STEllAR-GROUP/hpx/issues/4052
\(^{313}\) https://github.com/STEllAR-GROUP/hpx/issues/4032
\(^{314}\) https://github.com/STEllAR-GROUP/hpx/issues/4030
\(^{315}\) https://github.com/STEllAR-GROUP/hpx/issues/4029
\(^{316}\) https://github.com/STEllAR-GROUP/hpx/issues/4005
\(^{317}\) https://github.com/STEllAR-GROUP/hpx/issues/4002
\(^{318}\) https://github.com/STEllAR-GROUP/hpx/issues/3989
\(^{319}\) https://github.com/STEllAR-GROUP/hpx/issues/3958
\(^{320}\) https://github.com/STEllAR-GROUP/hpx/issues/3953
\(^{321}\) https://github.com/STEllAR-GROUP/hpx/issues/3941
\(^{322}\) https://github.com/STEllAR-GROUP/hpx/issues/3940
- Issue #3935 - HPX compiler match too strict?
- Issue #3929 - Buildbot failures on latest HPX stable
- Issue #3912 - I recommend publishing a version that does not depend on the boost library
- Issue #3896 - hpx.ini not working
- Issue #3883 - cuda compilation fails because of -faligned-new
- Issue #3879 - HPX fails to configure with -DHPX_WITH_TESTS=OFF
- Issue #3871 - dataflow does not support void allocators
- Issue #3867 - Latest HTML docs placed in wrong directory on GitHub pages
- Issue #3866 - Make sure all tests use HPX_TEST* macros and not HPX_ASSERT
- Issue #3857 - CMake all-keyword or all-plain for target_link_libraries
- Issue #3856 - hpx_setup_target adds rogue flags
- Issue #3850 - HPX fails to build on POWER8 with Clang7
- Issue #3848 - Remove lva member from thread_init_data
- Issue #3838 - hpx::parallel::count/count_if failing tests
- Issue #3651 - hpx::parallel::transform_reduce with non const reference as lambda parameter
- Issue #3560 - Apex integration with HPX not working properly
- Issue #3322 - No warning when mixing debug/release builds

Closed pull requests

- PR #4300 - Checks for MPI_Init being called twice
- PR #4299 - Small CMake fixes
- PR #4298 - Remove extra call to annotate function that messes up traces
- PR #4296 - Fixing collectives locking problem
- PR #4295 - Do not check LICENSE_1_0.txt for inspect violations
• PR #4293^345 - Applying two small changes fixing carious MSVC/Windows problems
• PR #4283^346 - Delete apex.hpp
• PR #4276^347 - Disable doxygen generation for hpx/debugging/print.hpp file
• PR #4275^348 - Make sure APEX is linked to even when not explicitly referenced
• PR #4272^349 - Fix pushing of documentation
• PR #4271^350 - Updating APEX tag, don’t create new task_wrapper on operator= of hpx_thread object
• PR #4268^351 - Testing for noexcept function specializations in in C++11/14 mode
• PR #4267^352 - Fixing MSVC warning
• PR #4266^353 - Make sure macOS Travis CI fails if build step fails
• PR #4264^354 - Clean up compatibility header options
• PR #4262^355 - Cleanup modules CMakeLists.txt
• PR #4261^356 - Fixing HPX/APEX linking and dependencies for external projects like Phylanx
• PR #4260^357 - Fix docs compilation problems
• PR #4258^358 - Couple of minor changes
• PR #4257^359 - Fix apex annotation for async dispatch
• PR #4256^360 - Remove lambdas from assert expressions
• PR #4255^361 - Ignoring lock in all_to_all and all_reduce
• PR #4254^362 - Adding action specializations for noexcept functions
• PR #4253^363 - Move partlit.hpp to affinity module
• PR #4252^364 - Make mismatching build types a hard error in CMake
• PR #4249^365 - Scheduler improvement
• PR #4248^366 - update hpxmp tag to v0.3.0
• PR #4245^367 - Adding high performance channels

^345 https://github.com/STEllAR-GROUP/hpx/pull/4293
^346 https://github.com/STEllAR-GROUP/hpx/pull/4285
^347 https://github.com/STEllAR-GROUP/hpx/pull/4276
^348 https://github.com/STEllAR-GROUP/hpx/pull/4275
^349 https://github.com/STEllAR-GROUP/hpx/pull/4271
^350 https://github.com/STEllAR-GROUP/hpx/pull/4268
^351 https://github.com/STEllAR-GROUP/hpx/pull/4266
^352 https://github.com/STEllAR-GROUP/hpx/pull/4264
^353 https://github.com/STEllAR-GROUP/hpx/pull/4262
^354 https://github.com/STEllAR-GROUP/hpx/pull/4261
^355 https://github.com/STEllAR-GROUP/hpx/pull/4260
^356 https://github.com/STEllAR-GROUP/hpx/pull/4258
^357 https://github.com/STEllAR-GROUP/hpx/pull/4257
^358 https://github.com/STEllAR-GROUP/hpx/pull/4256
^359 https://github.com/STEllAR-GROUP/hpx/pull/4255
^360 https://github.com/STEllAR-GROUP/hpx/pull/4254
^361 https://github.com/STEllAR-GROUP/hpx/pull/4253
^362 https://github.com/STEllAR-GROUP/hpx/pull/4252
^363 https://github.com/STEllAR-GROUP/hpx/pull/4249
^364 https://github.com/STEllAR-GROUP/hpx/pull/4248
^365 https://github.com/STEllAR-GROUP/hpx/pull/4245
- PR #4244 - Ignore lock in ignore_while_locked_1485 test
- PR #4243 - Fix PAPI command line option documentation
- PR #4242 - Ignore lock in target_distribution_policy
- PR #4241 - Fix start_stop_callbacks test
- PR #4240 - Mostly fix clang CUDA compilation
- PR #4238 - Google Season of Docs updates to documentation; grammar edits.
- PR #4237 - fixing annotated task to use the name, not the desc
- PR #4236 - Move module print summary to modules
- PR #4235 - Don’t use alignas in cache_{aligned, line}_data
- PR #4234 - Add basic overview sentence to all modules
- PR #4230 - Add OS X builds to Travis CI
- PR #4229 - Remove leftover queue compatibility checks
- PR #4226 - Fixing APEX shutdown by explicitly shutting down throttling
- PR #4225 - Allow CMAKE_INSTALL_PREFIX to be a relative path
- PR #4224 - Deprecate verbs parcelport
- PR #4222 - Update register_{thread, work} namespaces
- PR #4221 - Changing HPX_GCC_VERSION check from 70000 to 70300
- PR #4218 - Google Season of Docs updates to documentation; grammar edits.
- PR #4217 - Google Season of Docs updates to documentation; grammar edits.
- PR #4216 - Fixing gcc warning on 32bit platforms (integer truncation)
- PR #4214 - Apex callback refactoring
- PR #4213 - Clean up allocator checks for dependent projects
- PR #4212 - Google Season of Docs updates to documentation; grammar edits.

https://github.com/STEllAR-GROUP/hpx/pull/4244
https://github.com/STEllAR-GROUP/hpx/pull/4243
https://github.com/STEllAR-GROUP/hpx/pull/4242
https://github.com/STEllAR-GROUP/hpx/pull/4241
https://github.com/STEllAR-GROUP/hpx/pull/4240
https://github.com/STEllAR-GROUP/hpx/pull/4238
https://github.com/STEllAR-GROUP/hpx/pull/4237
https://github.com/STEllAR-GROUP/hpx/pull/4236
https://github.com/STEllAR-GROUP/hpx/pull/4235
https://github.com/STEllAR-GROUP/hpx/pull/4234
https://github.com/STEllAR-GROUP/hpx/pull/4230
https://github.com/STEllAR-GROUP/hpx/pull/4229
https://github.com/STEllAR-GROUP/hpx/pull/4226
https://github.com/STEllAR-GROUP/hpx/pull/4225
https://github.com/STEllAR-GROUP/hpx/pull/4224
https://github.com/STEllAR-GROUP/hpx/pull/4222
https://github.com/STEllAR-GROUP/hpx/pull/4221
https://github.com/STEllAR-GROUP/hpx/pull/4218
https://github.com/STEllAR-GROUP/hpx/pull/4217
https://github.com/STEllAR-GROUP/hpx/pull/4216
https://github.com/STEllAR-GROUP/hpx/pull/4214
https://github.com/STEllAR-GROUP/hpx/pull/4213
https://github.com/STEllAR-GROUP/hpx/pull/4212
- PR #4211 - Google Season of Docs updates to documentation; contributing to hpx
- PR #4210 - Attempting to fix Intel compilation
- PR #4209 - Fix CUDA 10 build
- PR #4205 - Making sure that differences in CMAKE_BUILD_TYPE are not reported on multi-configuration cmake generators
- PR #4203 - Deprecate Vc
- PR #4202 - Fix CUDA configuration
- PR #4200 - Making sure hpx_wrap is not passed on to linker on non-Linux systems
- PR #4198 - Fix execution_agent.cpp compilation with GCC 5
- PR #4197 - Remove deprecated options for 1.4.0 release
- PR #4196 - minor fixes for building on OSX Darwin
- PR #4195 - Use full clone on CircleCI for pushing stable tag
- PR #4193 - Add scheduling hints to hello_world_distributed
- PR #4192 - Set up CUDA in HPXConfig.cmake
- PR #4191 - Export allocators root variables
- PR #4190 - Don’t use constexpr in thread_data with GCC <= 6
- PR #4189 - Only use quick_exit if available
- PR #4188 - Google Season of Docs updates to documentation; writing single node hpx applications
- PR #4186 - correct vc to cuda in cuda cmake
- PR #4184 - Resetting some cached variables to make sure those are re-filled
- PR #4183 - Fix hpxcxx configuration
- PR #4181 - Rename base libraries var
- PR #4180 - Move header left behind earlier to plugin module
- PR #4179 - Moving zip_iterator and transform_iterator to iterator_support module
• PR #4178 - Move checkpointing support to its own module
• PR #4177 - Small const fix to basic_execution module
• PR #4176 - Add back HPX_LIBRARIES and friends to HPXConfig.cmake
• PR #4175 - Make Vc public and add it to HPXConfig.cmake
• PR #4173 - Wait for runtime to be running before returning from hpx::start
• PR #4172 - More protection against shutdown problems in error handling scenarios.
• PR #4171 - Ignore lock in condition_variable::wait
• PR #4170 - Adding APEX dependency to MPI parcelport
• PR #4168 - Adding utility include
• PR #4167 - Add a condition to setup the external libraries
• PR #4166 - Add an INTERNAL_FLAGS option to link to hpx_internal_flags
• PR #4165 - Forward HPX_* cmake cache variables to external projects
• PR #4164 - Affinity and batch environment modules
• PR #4162 - Handle quick exit
• PR #4160 - Using target_link_libraries for cmake versions >= 3.12
• PR #4159 - Make sure HPX_WITH_NATIVE_TLS is forwarded to dependent projects
• PR #4158 - Adding allocator imported target as a dependency of allocator module
• PR #4157 - Add hpx_memory as a dependency of parcelport plugins
• PR #4156 - Stackless coroutines now can refer to themselves (through get_self() and friends)
• PR #4154 - Added CMake policy CMP0060 for HPX applications.
• PR #4153 - Add header iomanip to tests and tool
• PR #4152 - Casting MPI tag value
• PR #4149 - Add back private m_desc member variable in program_options module
• PR #4147 - Resource partitioner and threadmanager modules
• PR #4146 - Google Season of Docs updates to documentation; creating hpx projects
• PR #4145 - Adding basic support for stackless threads
• PR #4144 - Exclude test_client_1950 from all target
• PR #4142 - Add a new thread_pool_executor
• PR #4140 - Google Season of Docs updates to documentation; why hpx
• PR #4139 - Remove runtime includes from coroutines module
• PR #4138 - Forking boost::intrusive_ptr and adding it as hpx::intrusive_ptr
• PR #4137 - Fixing TSS destruction
• PR #4136 - HPX.Compute modules
• PR #4133 - Fix block_executor
• PR #4131 - Applying fixes based on reports from PVS Studio
• PR #4130 - Adding missing header to build system
• PR #4129 - Fixing compilation if HPX_WITH_DATAPAR_VC is enabled
• PR #4128 - Renaming moveonly_any to unique_any
• PR #4126 - Attempt to fix basic_any constructor for gcc 7
• PR #4125 - Changing extra_archive_data implementation
• PR #4124 - Don’t link to Boost.System unless required
• PR #4122 - Add kernel launch helper utility (+saxpy demo) and merge in octotiger changes
• PR #4121 - Fixing migration test if networking is disabled.
• PR #4120 - Google Season of Docs updates to documentation; hpx build system v1
• PR #4119 - Making sure chunk_size and max_chunk are actually applied to parallel algorithms if specified
• PR #4117 - Make CircleCI formatting check store diff

https://github.com/STEllAR-GROUP/hpx/pull/4147
https://github.com/STEllAR-GROUP/hpx/pull/4146
https://github.com/STEllAR-GROUP/hpx/pull/4145
https://github.com/STEllAR-GROUP/hpx/pull/4143
https://github.com/STEllAR-GROUP/hpx/pull/4142
https://github.com/STEllAR-GROUP/hpx/pull/4140
https://github.com/STEllAR-GROUP/hpx/pull/4139
https://github.com/STEllAR-GROUP/hpx/pull/4138
https://github.com/STEllAR-GROUP/hpx/pull/4137
https://github.com/STEllAR-GROUP/hpx/pull/4136
https://github.com/STEllAR-GROUP/hpx/pull/4133
https://github.com/STEllAR-GROUP/hpx/pull/4131
https://github.com/STEllAR-GROUP/hpx/pull/4130
https://github.com/STEllAR-GROUP/hpx/pull/4129
https://github.com/STEllAR-GROUP/hpx/pull/4128
https://github.com/STEllAR-GROUP/hpx/pull/4126
https://github.com/STEllAR-GROUP/hpx/pull/4125
https://github.com/STEllAR-GROUP/hpx/pull/4124
https://github.com/STEllAR-GROUP/hpx/pull/4122
https://github.com/STEllAR-GROUP/hpx/pull/4121
https://github.com/STEllAR-GROUP/hpx/pull/4120
https://github.com/STEllAR-GROUP/hpx/pull/4119
https://github.com/STEllAR-GROUP/hpx/pull/4117
• PR #4116 - Fix automatically setting C++ standard
• PR #4114 - Module serialization
• PR #4113 - Module datastructures
• PR #4111 - Fixing performance regression introduced earlier
• PR #4110 - Adding missing SPDX tags
• PR #4109 - Overload for start without entry point/argv.
• PR #4108 - Making sure C++ standard is properly detected and propagated
• PR #4106 - use std::round for guaranteed rounding without errors
• PR #4104 - Extend scheduler_mode with new work_stealing and task assignment modes
• PR #4103 - Add this to lambda capture list
• PR #4102 - Add spdx license and check
• PR #4099 - Module coroutines
• PR #4098 - Fix append module path in module CMakeLists template
• PR #4097 - Function tests
• PR #4096 - Removing return of thread_result_type from functions not needing them
• PR #4095 - Stop-gap measure until cmake overhaul is in place
• PR #4094 - Deprecate HPX_WITH_MORE_THAN_64_THREADS
• PR #4093 - Fix initialization of global_num_tasks in parallel_executor
• PR #4092 - Add support for mi-malloc
• PR #4090 - Execution context
• PR #4089 - Make counters in coroutines optional
• PR #4087 - Making hpx::util::any compatible with C++17
• PR #4084 - Making sure destination array for std::transform is properly resized

https://github.com/STEllAR-GROUP/hpx/pull/4116
https://github.com/STEllAR-GROUP/hpx/pull/4114
https://github.com/STEllAR-GROUP/hpx/pull/4113
https://github.com/STEllAR-GROUP/hpx/pull/4111
https://github.com/STEllAR-GROUP/hpx/pull/4110
https://github.com/STEllAR-GROUP/hpx/pull/4109
https://github.com/STEllAR-GROUP/hpx/pull/4108
https://github.com/STEllAR-GROUP/hpx/pull/4106
https://github.com/STEllAR-GROUP/hpx/pull/4104
https://github.com/STEllAR-GROUP/hpx/pull/4103
https://github.com/STEllAR-GROUP/hpx/pull/4102
https://github.com/STEllAR-GROUP/hpx/pull/4099
https://github.com/STEllAR-GROUP/hpx/pull/4098
https://github.com/STEllAR-GROUP/hpx/pull/4097
https://github.com/STEllAR-GROUP/hpx/pull/4096
https://github.com/STEllAR-GROUP/hpx/pull/4095
https://github.com/STEllAR-GROUP/hpx/pull/4094
https://github.com/STEllAR-GROUP/hpx/pull/4093
https://github.com/STEllAR-GROUP/hpx/pull/4092
https://github.com/STEllAR-GROUP/hpx/pull/4090
https://github.com/STEllAR-GROUP/hpx/pull/4089
https://github.com/STEllAR-GROUP/hpx/pull/4087
https://github.com/STEllAR-GROUP/hpx/pull/4084
• PR #4083 - Adapting thread_queue_mc to behave even if no 128bit atomics are available
• PR #4082 - Fix compilation on GCC 5
• PR #4081 - Adding option allowing to force using Boost.FileSystem
• PR #4080 - Updating module dependencies
• PR #4079 - Add missing tests for iterator_support module
• PR #4078 - Disable parcel-layer if networking is disabled
• PR #4077 - Add missing include that causes build fails
• PR #4076 - Enable compatibility headers for functional module
• PR #4075 - Coroutines module
• PR #4073 - Use configure_file for generated files in modules
• PR #4071 - Fixing MPI detection for PMIx
• PR #4070 - Fix macOS builds
• PR #4069 - Moving more facilities to the collectives module
• PR #4068 - Adding main HPX #include directory to modules
• PR #4067 - Switching the use of message(STATUS "...") to hpx_info
• PR #4065 - Move Boost.Filesystem handling to filesystem module
• PR #4064 - Fix program_options test with older boost versions
• PR #4062 - The cpu_features tool fails to compile on anything but x86 architectures
• PR #4061 - Add clang-format checking step for modules
• PR #4060 - Making sure HPX_IDLE_BACKOFF_TIME_MAX is always defined (even if its unused)
• PR #4059 - Renaming module hpx_parallel_executors into hpx_execution
• PR #4058 - Do not build networking tests when networking disabled
• PR #4057 - Printing configuration summary for modules as well
• PR #4055 - Google Season of Docs updates to documentation; hpx build systems
• PR #4054 - Add troubleshooting section to manual
• PR #4051 - Add more variations to future_overhead test
• PR #4050 - Creating plugin module
• PR #4049 - Move missing modules tests
• PR #4047 - Add boost/filesystem headers to inspect deprecated headers
• PR #4045 - Module functional
• PR #4043 - Fix preconditions and error messages for suspension functions
• PR #4041 - Pass HPX_STANDARD on to dependent projects via HPXConfig.cmake
• PR #4040 - Program options module
• PR #4039 - Moving non-serializable any (any_nonser) to datastructures module
• PR #4038 - Adding MPark’s variant (V1.4.0) to HPX
• PR #4037 - Adding resiliency module
• PR #4036 - Add C++17 filesystem compatibility header
• PR #4035 - Fixing support for mpirun
• PR #4028 - CMake to target based directives
• PR #4027 - Remove GitLab CI configuration
• PR #4026 - Threading refactoring
• PR #4025 - Refactoring thread queue configuration options
• PR #4024 - Fix padding calculation in cache_aligned_data.hpp
• PR #4023 - Fixing Codacy issues
• PR #4022 - Make sure process mask option is passed to affinity_data
• PR #4021 - Warn about compiling in C++11 mode
• PR #4015 - Add more variations to future_overhead test
• PR #4020\(^{529}\) - Module concurrency
• PR #4016\(^{530}\) - Module topology
• PR #4018\(^{531}\) - Update deprecated header in thread_queue_mc.hpp
• PR #4015\(^{532}\) - Avoid overwriting artifacts
• PR #4014\(^{533}\) - Future overheads
• PR #4013\(^{534}\) - Update URL to test output conversion script
• PR #4012\(^{535}\) - Fix CUDA compilation
• PR #4011\(^{536}\) - Fixing cyclic dependencies between modules
• PR #4010\(^{537}\) - Ignore stable tag on CircleCI
• PR #4009\(^{538}\) - Check circular dependencies in a circle ci step
• PR #4008\(^{539}\) - Extend cache aligned data to handle tuple-like data
• PR #4007\(^{540}\) - Fixing migration for components that have actions returning a client
• PR #4006\(^{541}\) - Move is_value_proxy.hpp to algorithms module
• PR #4004\(^{542}\) - Shorten CTest timeout on CircleCI
• PR #4003\(^{543}\) - Refactoring to remove (internal) dependencies
• PR #4001\(^{544}\) - Exclude tests from all target
• PR #4000\(^{545}\) - Module errors
• PR #3999\(^{546}\) - Enable support for compatibility headers for logging module
• PR #3998\(^{547}\) - Add process thread binding option
• PR #3997\(^{548}\) - Export handle_assert function
• PR #3996\(^{549}\) - Attempt to solve issue where -latomic does not support 128bit atomics
• PR #3993\(^{550}\) - Make sure __LINE__ is an unsigned
• PR #3991\(^{551}\) - Fix dependencies and flags for for header tests

\(^{529}\) https://github.com/STEllAR-GROUP/hpx/pull/4020
\(^{530}\) https://github.com/STEllAR-GROUP/hpx/pull/4019
\(^{531}\) https://github.com/STEllAR-GROUP/hpx/pull/4018
\(^{532}\) https://github.com/STEllAR-GROUP/hpx/pull/4015
\(^{533}\) https://github.com/STEllAR-GROUP/hpx/pull/4014
\(^{534}\) https://github.com/STEllAR-GROUP/hpx/pull/4013
\(^{535}\) https://github.com/STEllAR-GROUP/hpx/pull/4012
\(^{536}\) https://github.com/STEllAR-GROUP/hpx/pull/4011
\(^{537}\) https://github.com/STEllAR-GROUP/hpx/pull/4010
\(^{538}\) https://github.com/STEllAR-GROUP/hpx/pull/4009
\(^{539}\) https://github.com/STEllAR-GROUP/hpx/pull/4008
\(^{540}\) https://github.com/STEllAR-GROUP/hpx/pull/4007
\(^{541}\) https://github.com/STEllAR-GROUP/hpx/pull/4006
\(^{542}\) https://github.com/STEllAR-GROUP/hpx/pull/4004
\(^{543}\) https://github.com/STEllAR-GROUP/hpx/pull/4003
\(^{544}\) https://github.com/STEllAR-GROUP/hpx/pull/4001
\(^{545}\) https://github.com/STEllAR-GROUP/hpx/pull/4000
\(^{546}\) https://github.com/STEllAR-GROUP/hpx/pull/3999
\(^{547}\) https://github.com/STEllAR-GROUP/hpx/pull/3998
\(^{548}\) https://github.com/STEllAR-GROUP/hpx/pull/3997
\(^{549}\) https://github.com/STEllAR-GROUP/hpx/pull/3996
\(^{550}\) https://github.com/STEllAR-GROUP/hpx/pull/3993
\(^{551}\) https://github.com/STEllAR-GROUP/hpx/pull/3991
• PR #3990\(^552\) - Documentation tags fixes
• PR #3985\(^553\) - Adding missing solution folder for format module test
• PR #3987\(^554\) - Move runtime-dependent functions out of command line handling
• PR #3986\(^555\) - Fix CMake configuration with PAPI on
• PR #3985\(^556\) - Module timing
• PR #3984\(^557\) - Fix default behaviour of paths in \texttt{add_hpx\_component}
• PR #3982\(^558\) - Parallel executors module
• PR #3981\(^559\) - Segmented algorithms module
• PR #3980\(^560\) - Module logging
• PR #3979\(^561\) - Module util
• PR #3978\(^562\) - Fix \texttt{clang\_tidy} step on CircleCI
• PR #3977\(^563\) - Fixing solution folders for moved components
• PR #3976\(^564\) - Module format
• PR #3975\(^565\) - Enable deprecation warnings on CircleCI
• PR #3974\(^566\) - Fix typos in documentation
• PR #3973\(^567\) - Fix compilation with GCC 9
• PR #3972\(^568\) - Add condition to clone apex + use of new cmake var APEX\_ROOT
• PR #3971\(^569\) - Add testing module
• PR #3968\(^570\) - Remove unneeded file in hardware module
• PR #3967\(^571\) - Remove leftover PIC settings from main CMakeLists.txt
• PR #3966\(^572\) - Add missing export option in \texttt{add_hpx\_module}
• PR #3965\(^573\) - Change \texttt{current\_function\_helper} back to \texttt{non-constexpr}
• PR #3964\(^574\) - Fixing merge problems

\(^552\) https://github.com/STEllAR-GROUP/hpx/pull/3990
\(^553\) https://github.com/STEllAR-GROUP/hpx/pull/3988
\(^554\) https://github.com/STEllAR-GROUP/hpx/pull/3987
\(^555\) https://github.com/STEllAR-GROUP/hpx/pull/3986
\(^556\) https://github.com/STEllAR-GROUP/hpx/pull/3985
\(^557\) https://github.com/STEllAR-GROUP/hpx/pull/3984
\(^558\) https://github.com/STEllAR-GROUP/hpx/pull/3982
\(^559\) https://github.com/STEllAR-GROUP/hpx/pull/3981
\(^560\) https://github.com/STEllAR-GROUP/hpx/pull/3980
\(^561\) https://github.com/STEllAR-GROUP/hpx/pull/3979
\(^562\) https://github.com/STEllAR-GROUP/hpx/pull/3978
\(^563\) https://github.com/STEllAR-GROUP/hpx/pull/3977
\(^564\) https://github.com/STEllAR-GROUP/hpx/pull/3976
\(^565\) https://github.com/STEllAR-GROUP/hpx/pull/3975
\(^566\) https://github.com/STEllAR-GROUP/hpx/pull/3974
\(^567\) https://github.com/STEllAR-GROUP/hpx/pull/3973
\(^568\) https://github.com/STEllAR-GROUP/hpx/pull/3972
\(^569\) https://github.com/STEllAR-GROUP/hpx/pull/3971
\(^570\) https://github.com/STEllAR-GROUP/hpx/pull/3968
\(^571\) https://github.com/STEllAR-GROUP/hpx/pull/3967
\(^572\) https://github.com/STEllAR-GROUP/hpx/pull/3966
\(^573\) https://github.com/STEllAR-GROUP/hpx/pull/3965
\(^574\) https://github.com/STEllAR-GROUP/hpx/pull/3964
- PR #3962 - Add a trait for std::array for unwrapping
- PR #3961 - Making hpx::util::tuple<Ts...> and std::tuple<Ts...> convertible
- PR #3960 - Fix compilation with CUDA 10 and GCC 6
- PR #3958 - Fix C++11 incompatibility
- PR #3957 - Algorithms module
- PR #3956 - [HPX_AddModule] Fix lower name var to upper
- PR #3955 - Fix CMake configuration with examples off and tests on
- PR #3954 - Move components to separate subdirectory in root of repository
- PR #3952 - Update papi.cpp
- PR #3951 - Exclude modules header tests from all target
- PR #3950 - Adding all_reduce facility to collectives module
- PR #3949 - This adds a configuration file that will cause for stale issues to be automatically closed
- PR #3948 - Fixing ALPS environment
- PR #3947 - Add major compiler version check for building hpx as a binary package
- PR #3946 - [Modules] Move the location of the generated headers
- PR #3945 - Simplify tests and examples cmake
- PR #3943 - Remove example module
- PR #3942 - Add NOEXPORT option to add_hpx_(component,library)
- PR #3938 - Use https for CDash submissions
- PR #3937 - Add HPX_WITH_BUILD_BINARY_PACKAGE to the compiler check (refs #3935)
- PR #3936 - Fixing installation of binaries on windows
- PR #3934 - Add set function for sliding_semaphore max_difference
- PR #3933 - Remove cudadevrt from compile/link flags as it breaks downstream projects

575 https://github.com/STEllAR-GROUP/hpx/pull/3962
576 https://github.com/STEllAR-GROUP/hpx/pull/3961
577 https://github.com/STEllAR-GROUP/hpx/pull/3960
578 https://github.com/STEllAR-GROUP/hpx/pull/3959
579 https://github.com/STEllAR-GROUP/hpx/pull/3957
580 https://github.com/STEllAR-GROUP/hpx/pull/3956
581 https://github.com/STEllAR-GROUP/hpx/pull/3955
582 https://github.com/STEllAR-GROUP/hpx/pull/3954
583 https://github.com/STEllAR-GROUP/hpx/pull/3952
584 https://github.com/STEllAR-GROUP/hpx/pull/3951
585 https://github.com/STEllAR-GROUP/hpx/pull/3950
586 https://github.com/STEllAR-GROUP/hpx/pull/3949
587 https://github.com/STEllAR-GROUP/hpx/pull/3948
588 https://github.com/STEllAR-GROUP/hpx/pull/3947
589 https://github.com/STEllAR-GROUP/hpx/pull/3946
590 https://github.com/STEllAR-GROUP/hpx/pull/3945
591 https://github.com/STEllAR-GROUP/hpx/pull/3943
592 https://github.com/STEllAR-GROUP/hpx/pull/3942
593 https://github.com/STEllAR-GROUP/hpx/pull/3938
594 https://github.com/STEllAR-GROUP/hpx/pull/3937
595 https://github.com/STEllAR-GROUP/hpx/pull/3936
596 https://github.com/STEllAR-GROUP/hpx/pull/3934
597 https://github.com/STEllAR-GROUP/hpx/pull/3933
- PR #3932\(^{598}\) - Fixing 3929
- PR #3931\(^{599}\) - Adding \texttt{all\_to\_all}
- PR #3930\(^{600}\) - Add test demonstrating the use of broadcast with component actions
- PR #3928\(^{601}\) - fixed number of tasks and number of threads for heterogeneous slurm environments
- PR #3927\(^{602}\) - Moving Cache module’s tests into separate solution folder
- PR #3926\(^{603}\) - Move unit tests to cache module
- PR #3925\(^{604}\) - Move version check to config module
- PR #3924\(^{605}\) - Add schedule hint executor parameters
- PR #3923\(^{606}\) - Allow aligning objects bigger than the cache line size
- PR #3922\(^{607}\) - Add Windows builds with Travis CI
- PR #3921\(^{608}\) - Add ccl cache directory to gitignore
- PR #3920\(^{609}\) - Fix \texttt{git\_external} fetching of tags
- PR #3905\(^{610}\) - Correct rostambod url. Fix typo in doc
- PR #3904\(^{611}\) - Fix bug in \texttt{context\_base.hpp}
- PR #3903\(^{612}\) - Adding new performance counters
- PR #3902\(^{613}\) - Add \texttt{add\_hpx\_module\ function}
- PR #3901\(^{614}\) - Factoring out container remapping into a separate trait
- PR #3900\(^{615}\) - Making sure errors during command line processing are properly reported and will not cause assertions
- PR #3899\(^{616}\) - Remove old compatibility bases from \texttt{make\_action}
- PR #3898\(^{617}\) - Make parameter size be of type \texttt{size\_t}
- PR #3897\(^{618}\) - Making sure all tests are disabled if \texttt{HPX\_WITH\_TESTS=OFF}
- PR #3895\(^{619}\) - Add documentation for \texttt{annotated\_function}
- PR #3894\(^{620}\) - Working around VS2019 problem with \texttt{make\_action}

\(^{598}\) https://github.com/STEllAR-GROUP/hpx/pull/3932
\(^{599}\) https://github.com/STEllAR-GROUP/hpx/pull/3931
\(^{600}\) https://github.com/STEllAR-GROUP/hpx/pull/3930
\(^{601}\) https://github.com/STEllAR-GROUP/hpx/pull/3928
\(^{602}\) https://github.com/STEllAR-GROUP/hpx/pull/3927
\(^{603}\) https://github.com/STEllAR-GROUP/hpx/pull/3926
\(^{604}\) https://github.com/STEllAR-GROUP/hpx/pull/3925
\(^{605}\) https://github.com/STEllAR-GROUP/hpx/pull/3924
\(^{606}\) https://github.com/STEllAR-GROUP/hpx/pull/3923
\(^{607}\) https://github.com/STEllAR-GROUP/hpx/pull/3922
\(^{608}\) https://github.com/STEllAR-GROUP/hpx/pull/3921
\(^{609}\) https://github.com/STEllAR-GROUP/hpx/pull/3920
\(^{610}\) https://github.com/STEllAR-GROUP/hpx/pull/3905
\(^{611}\) https://github.com/STEllAR-GROUP/hpx/pull/3904
\(^{612}\) https://github.com/STEllAR-GROUP/hpx/pull/3903
\(^{613}\) https://github.com/STEllAR-GROUP/hpx/pull/3902
\(^{614}\) https://github.com/STEllAR-GROUP/hpx/pull/3901
\(^{615}\) https://github.com/STEllAR-GROUP/hpx/pull/3900
\(^{616}\) https://github.com/STEllAR-GROUP/hpx/pull/3899
\(^{617}\) https://github.com/STEllAR-GROUP/hpx/pull/3898
\(^{618}\) https://github.com/STEllAR-GROUP/hpx/pull/3897
\(^{619}\) https://github.com/STEllAR-GROUP/hpx/pull/3895
\(^{620}\) https://github.com/STEllAR-GROUP/hpx/pull/3894
• PR #3892 - Avoid MSVC compatibility warning in internal allocator
• PR #3891 - Removal of the default intel config include
• PR #3888 - Fix `async_customization` dataflow example and Clarify what's being tested
• PR #3876 - Add Doxygen documentation
• PR #3882 - Minor docs fixes
• PR #3880 - Updating APEX version tag
• PR #3878 - Making sure symbols are properly exported from modules (needed for Windows/MacOS)
• PR #3877 - Documentation
• PR #3876 - Module hardware
• PR #3875 - Converted typedefs in actions submodule to using directives
• PR #3874 - Allow to suppress target keywords in `hpx_setup_target` for backwards compatibility
• PR #3873 - Add scripts to create releases and generate lists of PRs and issues
• PR #3872 - Fix latest HTML docs location
• PR #3870 - Module cache
• PR #3869 - Post 1.3.0 version bumps
• PR #3868 - Replace the macro `HPX_ASSERT` by `HPX_TEST` in tests
• PR #3845 - Assertion module
• PR #3839 - Make tuple serialization non-intrusive
• PR #3832 - Config module
• PR #3799 - Remove compat namespace and its contents
• PR #3701 - MoodyCamel lockfree
• PR #3496 - Disabling MPI's (deprecated) C++ interface
• PR #3192 - Move type info into `hpx::debug` namespace and add print helper functions

621 https://github.com/STEllAR-GROUP/hpx/pull/3892
622 https://github.com/STEllAR-GROUP/hpx/pull/3891
623 https://github.com/STEllAR-GROUP/hpx/pull/3888
624 https://github.com/STEllAR-GROUP/hpx/pull/3887
625 https://github.com/STEllAR-GROUP/hpx/pull/3882
626 https://github.com/STEllAR-GROUP/hpx/pull/3880
627 https://github.com/STEllAR-GROUP/hpx/pull/3878
628 https://github.com/STEllAR-GROUP/hpx/pull/3877
629 https://github.com/STEllAR-GROUP/hpx/pull/3876
630 https://github.com/STEllAR-GROUP/hpx/pull/3875
631 https://github.com/STEllAR-GROUP/hpx/pull/3874
632 https://github.com/STEllAR-GROUP/hpx/pull/3873
633 https://github.com/STEllAR-GROUP/hpx/pull/3872
634 https://github.com/STEllAR-GROUP/hpx/pull/3870
635 https://github.com/STEllAR-GROUP/hpx/pull/3869
636 https://github.com/STEllAR-GROUP/hpx/pull/3868
637 https://github.com/STEllAR-GROUP/hpx/pull/3865
638 https://github.com/STEllAR-GROUP/hpx/pull/3859
639 https://github.com/STEllAR-GROUP/hpx/pull/3852
640 https://github.com/STEllAR-GROUP/hpx/pull/3799
641 https://github.com/STEllAR-GROUP/hpx/pull/3701
642 https://github.com/STEllAR-GROUP/hpx/pull/3496
643 https://github.com/STEllAR-GROUP/hpx/pull/3192
• PR #3159 - Support Checkpointing Components

2.11.3 HPX V1.3.0 (May 23, 2019)

General changes

• Performance improvements: the schedulers have significantly reduced overheads from removing false sharing and the parallel executor has been updated to create fewer futures.

• HPX now defaults to not turning on networking when running on one locality. This means that you can run multiple instances on the same system without adding command line options.

• Multiple issues reported by Clang sanitizers have been fixed.

• We have added (back) single-page HTML documentation and PDF documentation.

• We have started modularizing the HPX library. This is useful both for developers and users. In the long term users will be able to consume only parts of the HPX libraries if they do not require all the functionality that HPX currently provides.

• We have added an implementation of function_ref.

• The barrier and latch classes have gained a few additional member functions.

Breaking changes

• Executable and library targets are now created without the _exe and _lib suffix respectively. For example, the target ld_stencil_1_exe is now simply called ld_stencil_1.

• We have removed the following deprecated functionality: queue, scoped_unlock, and support for input iterators in algorithms.

• We have turned off the compatibility layer for unwrapped by default. The functionality will be removed in the next release. The option can still be turned on using the CMake option HPX_WITH_UNWRAPPED_SUPPORT. Likewise, inclusive_scan compatibility overloads have been turned off by default. They can still be turned on with HPX_WITH_INCLUSIVE_SCAN_COMPATIBILITY.

• The minimum compiler and dependency versions have been updated. We now support GCC from version 5 onwards, Clang from version 4 onwards, and Boost from version 1.61.0 onwards.

• The headers for preprocessor macros have moved as a result of the functionality being moved to a separate module. The old headers are deprecated and will be removed in a future version of HPX. You can turn off the warnings by setting HPX_PREPROCESSOR_WITH_DEPRECATED_WARNINGS=OFF or turn off the compatibility headers completely with HPX_PREPROCESSOR_WITH_COMPATIBILITY_HEADERS=OFF.

Closed issues

• Issue #3863 - shouldn’t “-faligned-new” be a usage requirement?

• Issue #3841 - Build error with msvc 19 caused by SFINAE and C++17

• Issue #3836 - master branch does not build with idle rate counters enabled

• Issue #3819 - Add debug suffix to modules built in debug mode

• Issue #3817 - HPX_INCLUDE_DIRS contains non-existent directory

644 https://github.com/STEllAR-GROUP/hpx/pull/3159
645 https://www.cmake.org
646 https://github.com/STEllAR-GROUP/hpx/issues/3863
647 https://github.com/STEllAR-GROUP/hpx/issues/3841
648 https://github.com/STEllAR-GROUP/hpx/issues/3836
649 https://github.com/STEllAR-GROUP/hpx/issues/3819
650 https://github.com/STEllAR-GROUP/hpx/issues/3817
• Issue #3810 - Source groups are not created for files in modules
• Issue #3805 - HPX won’t compile with -DHPX_WITH_APEX=TRUE
• Issue #3792 - Barrier Hangs When Locality Zero not included
• Issue #3778 - Replace throw() with noexcept
• Issue #3763 - configurable sort limit per task
• Issue #3758 - dataflow doesn’t convert future<future<T>> to future<T>
• Issue #3757 - When compiling undefined reference to hpx::hpx_check_version_1_2 HPX V1.2.1, Ubuntu 18.04.01 Server Edition
• Issue #3753 - --hpx:list-counters=full crashes
• Issue #3746 - Detection of MPI with pmix
• Issue #3744 - Separate spinlock from same cacheline as internal data for all LCOs
• Issue #3743 - hpxxxx’s shebang doesn’t specify the python version
• Issue #3738 - Unable to debug parcelport on a single node
• Issue #3735 - Latest master: Can’t compile in MSVC
• Issue #3731 - util::bound seems broken on Clang with older libstdc++
• Issue #3724 - Allow to pre-set command line options through environment
• Issue #3723 - examples/resource_partitioner build issue on master branch / ubuntu 18
• Issue #3721 - faced a building error
• Issue #3720 - Hello World example fails to link
• Issue #3719 - pkg-config produces invalid output: -l-pthread
• Issue #3718 - Please make the python executable configurable through cmake
• Issue #3717 - interested to contribute to the organisation
• Issue #3699 - Remove ‘HPX runtime’ executable
• Issue #3698 - Ignore all locks while handling asserts

651 https://github.com/STEllAR-GROUP/hpx/issues/3810
652 https://github.com/STEllAR-GROUP/hpx/issues/3805
653 https://github.com/STEllAR-GROUP/hpx/issues/3792
654 https://github.com/STEllAR-GROUP/hpx/issues/3778
655 https://github.com/STEllAR-GROUP/hpx/issues/3763
656 https://github.com/STEllAR-GROUP/hpx/issues/3758
657 https://github.com/STEllAR-GROUP/hpx/issues/3757
658 https://github.com/STEllAR-GROUP/hpx/issues/3753
659 https://github.com/STEllAR-GROUP/hpx/issues/3746
660 https://github.com/STEllAR-GROUP/hpx/issues/3744
661 https://github.com/STEllAR-GROUP/hpx/issues/3743
662 https://github.com/STEllAR-GROUP/hpx/issues/3738
663 https://github.com/STEllAR-GROUP/hpx/issues/3735
664 https://github.com/STEllAR-GROUP/hpx/issues/3731
665 https://github.com/STEllAR-GROUP/hpx/issues/3724
666 https://github.com/STEllAR-GROUP/hpx/issues/3723
667 https://github.com/STEllAR-GROUP/hpx/issues/3721
668 https://github.com/STEllAR-GROUP/hpx/issues/3720
669 https://github.com/STEllAR-GROUP/hpx/issues/3719
670 https://github.com/STEllAR-GROUP/hpx/issues/3718
671 https://github.com/STEllAR-GROUP/hpx/issues/3717
672 https://github.com/STEllAR-GROUP/hpx/issues/3699
673 https://github.com/STEllAR-GROUP/hpx/issues/3698

888  Chapter 2. What’s so special about HPX?
• Issue #3689 - Incorrect and inconsistent website structure http://stellar.cct.lsu.edu/downloads/.
• Issue #3681 - Broken links on http://stellar.cct.lsu.edu/2015/05/hpx-archives-now-on-gmane/
• Issue #3676 - HPX master built from source, cmake fails to link main.cpp example in docs
• Issue #3673 - HPX build fails with std::atomic missing error
• Issue #3670 - Generate PDF again from documentation (with Sphinx)
• Issue #3643 - Warnings when compiling HPX 1.2.1 with gcc 9
• Issue #3641 - Trouble with using ranges-v3 and hpx::parallel::reduce
• Issue #3639 - util::unwrapping does not work well with member functions
• Issue #3634 - The build fails if shared_future<>::then is called with a thread executor
• Issue #3632 - VTune Amplifier 2019 not working with use_itt_notify=1
• Issue #3616 - HPX Fails to Build with CUDA 10
• Issue #3612 - False sharing of scheduling counters
• Issue #3609 - executor_parameters timeout with gcc <= 7 and Debug mode
• Issue #3601 - Missleading error message on power pc for rdtsc and rdtscp
• Issue #3598 - Build of some examples fails when using Vc
• Issue #3594 - Error: The number of OS threads requested (20) does not match the number of threads to bind (12): HPX(bad_parameter)
• Issue #3592 - Undefined Reference Error
• Issue #3589 - include could not find load file: HPX_Utils.cmake
• Issue #3587 - HPX won’t compile on POWER8 with Clang 7
• Issue #3583 - Fedora and openSUSE instructions missing on “Distribution Packages” page
• Issue #3578 - Build error when configuring with HPX_HAVE_ALGORITHM_INPUT_ITERATOR_SUPPORT=ON
• Issue #3575 - Merge openSUSE reproducible patch
• Issue #3570 - Update HPX to work with the latest VC version

674 https://github.com/STEllAR-GROUP/hpx/issues/3689
675 https://github.com/STEllAR-GROUP/hpx/issues/3681
676 https://github.com/STEllAR-GROUP/hpx/issues/3676
677 https://github.com/STEllAR-GROUP/hpx/issues/3673
678 https://github.com/STEllAR-GROUP/hpx/issues/3670
679 https://github.com/STEllAR-GROUP/hpx/issues/3643
680 https://github.com/STEllAR-GROUP/hpx/issues/3641
681 https://github.com/STEllAR-GROUP/hpx/issues/3639
682 https://github.com/STEllAR-GROUP/hpx/issues/3634
683 https://github.com/STEllAR-GROUP/hpx/issues/3622
684 https://github.com/STEllAR-GROUP/hpx/issues/3616
685 https://github.com/STEllAR-GROUP/hpx/issues/3612
686 https://github.com/STEllAR-GROUP/hpx/issues/3609
687 https://github.com/STEllAR-GROUP/hpx/issues/3601
688 https://github.com/STEllAR-GROUP/hpx/issues/3598
689 https://github.com/STEllAR-GROUP/hpx/issues/3594
690 https://github.com/STEllAR-GROUP/hpx/issues/3592
691 https://github.com/STEllAR-GROUP/hpx/issues/3589
692 https://github.com/STEllAR-GROUP/hpx/issues/3587
693 https://github.com/STEllAR-GROUP/hpx/issues/3583
694 https://github.com/STEllAR-GROUP/hpx/issues/3578
695 https://github.com/STEllAR-GROUP/hpx/issues/3575
696 https://github.com/STEllAR-GROUP/hpx/issues/3570
• Issue #3567 - Build succeed and make failed for hpx:cout
• Issue #3565 - Polymorphic simple component destructor not getting called
• Issue #3559 - 1.2.0 is missing from download page
• Issue #3554 - Clang 6.0 warning of hiding overloaded virtual function
• Issue #3510 - Build on ppc64 fails
• Issue #3482 - Improve error message when HPX_WITH_MAX_CPU_COUNT is too low for given system
• Issue #3453 - Two HPX applications can’t run at the same time.
• Issue #3452 - Scaling issue on the change to 2 NUMA domains
• Issue #3442 - HPX set_difference, set_intersection failure cases
• Issue #3437 - Ensure parent_task pointer when child task is created and child/parent are on same locality
• Issue #3255 - Suspension with lock for --hpx:list-component-types
• Issue #3034 - Use C++17 structured bindings for serialization
• Issue #2999 - Change thread scheduling use of size_t for thread indexing

Closed pull requests

• PR #3865 - adds hpx_target_compile_option_if_available
• PR #3864 - Helper functions that are useful in numa binding and testing of allocator
• PR #3862 - Temporary fix to local_dataflow_boost_small_vector test
• PR #3860 - Add cache line padding to intermediate results in for loop reduction
• PR #3859 - Remove HPX_TLL_PUBLIC and HPX_TLL_PRIVATE from CMake files
• PR #3858 - Add compile flags and definitions to modules
• PR #3851 - update hpxmp release tag to v0.2.0
• PR #3849 - Correct BOOST_ROOT variable name in quick start guide
• PR #3847 - Fix attach_debugger configuration option
• PR #3846 - Add tests for libs header tests
• PR #3844 - Fixing source_groups in preprocessor module to properly handle compatibility headers
• PR #3843 - This fixes the launch_process/launched_process pair of tests
• PR #3842 - Fix macro call with ITTNOTIFY enabled
• PR #3840 - Fixing SLURM environment parsing
• PR #3837 - Fixing misplaced #endif
• PR #3835 - make all latch members protected for consistency
• PR #3834 - Disable transpose_block_numa example on CircleCI
• PR #3833 - make latch counter protected for deriving latch in hpxmp
• PR #3831 - Fix CircleCI config for modules
• PR #3830 - minor fix: option HPX_WITH_TEST was not working correctly
• PR #3828 - Avoid for binaries that depend on HPX to directly link against internal modules
• PR #3827 - Adding shortcut for hpx::get_ptr<>(sync, id) for a local, non-migratable objects
• PR #3826 - Fix and update modules documentation
• PR #3825 - Updating default APEX version to 2.1.3 with HPX
• PR #3823 - Fix pkgconfig libs handling
• PR #3822 - Change includes in hpx_wrap.cpp to more specific includes
• PR #3821 - Disable barrier_3792 test when networking is disabled
• PR #3820 - Assorted CMake fixes
• PR #3815 - Removing left-over debug output
• PR #3814 - Allow setting default scheduler mode via the configuration database
• PR #3813 - Make the deprecation warnings issued by the old pp headers optional
• PR #3812 - Windows requires to handle symlinks to directories differently from those linking files
- PR #3811 - Clean up PP module and library skeleton
- PR #3806 - Moving include path configuration to before APEX
- PR #3804 - Fix latch
- PR #3803 - Update hpxcxx to look at lib64 and use python3
- PR #3802 - Numa binding allocator
- PR #3801 - Remove duplicated includes
- PR #3800 - Attempt to fix Posix context switching after lazy init changes
- PR #3798 - count and count_if accepts different iterator types
- PR #3797 - Adding a couple of override keywords to overloaded virtual functions
- PR #3796 - Re-enable testing all schedulers in shutdown_suspended_test
- PR #3795 - Change std::terminate to std::abort in SIGSEGV handler
- PR #3794 - Fixing #3792
- PR #3793 - Extending migrate_polymorphic_component unit test
- PR #3791 - Change throw() to noexcept
- PR #3790 - Remove deprecated options for 1.3.0 release
- PR #3789 - Remove Boost filesystem compatibility header
- PR #3788 - Disabled even more spots that should not execute if networking is disabled
- PR #3787 - Bump minimal boost supported version to 1.61.0
- PR #3786 - Bump minimum required versions for 1.3.0 release
- PR #3785 - Explicitly set number of jobs for all ninja invocations on CircleCI
- PR #3784 - Fix leak and address sanitizer problems
- PR #3783 - Disabled even more spots that should not execute is networking is disabled
- PR #3782 - Cherry-picked tuple and thread_init_data fixes from #3701

[742](https://github.com/STEllAR-GROUP/hpx/pull/3811)
[743](https://github.com/STEllAR-GROUP/hpx/pull/3806)
[744](https://github.com/STEllAR-GROUP/hpx/pull/3804)
[745](https://github.com/STEllAR-GROUP/hpx/pull/3803)
[746](https://github.com/STEllAR-GROUP/hpx/pull/3802)
[747](https://github.com/STEllAR-GROUP/hpx/pull/3801)
[748](https://github.com/STEllAR-GROUP/hpx/pull/3800)
[749](https://github.com/STEllAR-GROUP/hpx/pull/3798)
[750](https://github.com/STEllAR-GROUP/hpx/pull/3797)
[751](https://github.com/STEllAR-GROUP/hpx/pull/3796)
[752](https://github.com/STEllAR-GROUP/hpx/pull/3795)
[753](https://github.com/STEllAR-GROUP/hpx/pull/3794)
[754](https://github.com/STEllAR-GROUP/hpx/pull/3793)
[755](https://github.com/STEllAR-GROUP/hpx/pull/3792)
[756](https://github.com/STEllAR-GROUP/hpx/pull/3791)
[757](https://github.com/STEllAR-GROUP/hpx/pull/3790)
[758](https://github.com/STEllAR-GROUP/hpx/pull/3789)
[759](https://github.com/STEllAR-GROUP/hpx/pull/3788)
[760](https://github.com/STEllAR-GROUP/hpx/pull/3787)
[761](https://github.com/STEllAR-GROUP/hpx/pull/3786)
[762](https://github.com/STEllAR-GROUP/hpx/pull/3785)
[763](https://github.com/STEllAR-GROUP/hpx/pull/3784)
[764](https://github.com/STEllAR-GROUP/hpx/pull/3783)
• PR #3781 - Fix generic context coroutines after lazy stack allocation changes
• PR #3780 - Rename hello world examples
• PR #3776 - Sort algorithms now use the supplied chunker to determine the required minimal chunk size
• PR #3775 - Disable Boost auto-linking
• PR #3774 - Tag and push stable builds
• PR #3773 - Enable migration of polymorphic components
• PR #3771 - Fix link to stackoverflow in documentation
• PR #3770 - Replacing constexpr if in brace-serialization code
• PR #3769 - Fix SIGSEGV handler
• PR #3768 - Adding flags to scheduler allowing to control thread stealing and idle back-off
• PR #3767 - Fix help formatting in hpxrun.py
• PR #3765 - Fix a couple of bugs in the thread test
• PR #3764 - Workaround for SFINAE regression in msvc14.2
• PR #3762 - Prevent MSVC from prematurely instantiating things
• PR #3761 - Update python scripts to work with python 3
• PR #3760 - Fix callable vtable for GCC4.9
• PR #3759 - Rename PAGE_SIZE to PAGE_SIZE_ because AppleClang
• PR #3758 - Making sure locks are not held during suspension
• PR #3754 - Disable more code if networking is not available/not enabled
• PR #3752 - Move util::format implementation to source file
• PR #3751 - Fixing problems with lcos::barrier and iostreams
• PR #3750 - Change error message to take into account use_guard_page setting
• PR #3749 - Fix lifetime problem in run_as_hpx_thread

https://github.com/STEllAR-GROUP/hpx/pull/3781
https://github.com/STEllAR-GROUP/hpx/pull/3780
https://github.com/STEllAR-GROUP/hpx/pull/3776
https://github.com/STEllAR-GROUP/hpx/pull/3775
https://github.com/STEllAR-GROUP/hpx/pull/3774
https://github.com/STEllAR-GROUP/hpx/pull/3773
https://github.com/STEllAR-GROUP/hpx/pull/3771
https://github.com/STEllAR-GROUP/hpx/pull/3770
https://github.com/STEllAR-GROUP/hpx/pull/3769
https://github.com/STEllAR-GROUP/hpx/pull/3768
https://github.com/STEllAR-GROUP/hpx/pull/3767
https://github.com/STEllAR-GROUP/hpx/pull/3765
https://github.com/STEllAR-GROUP/hpx/pull/3764
https://github.com/STEllAR-GROUP/hpx/pull/3762
https://github.com/STEllAR-GROUP/hpx/pull/3761
https://github.com/STEllAR-GROUP/hpx/pull/3760
https://github.com/STEllAR-GROUP/hpx/pull/3759
https://github.com/STEllAR-GROUP/hpx/pull/3755
https://github.com/STEllAR-GROUP/hpx/pull/3754
https://github.com/STEllAR-GROUP/hpx/pull/3752
https://github.com/STEllAR-GROUP/hpx/pull/3751
https://github.com/STEllAR-GROUP/hpx/pull/3750
https://github.com/STEllAR-GROUP/hpx/pull/3749
• PR #3748 - Fixed unusable behavior of the clang code analyzer.
• PR #3747 - Added PMIX_RANK to the defaults of HPX_WITH_PARCELPORT_MPI_ENV.
• PR #3745 - Introduced cache_aligned_data and cache_line_data helper structure.
• PR #3742 - Remove more unused functionality from util/logging.
• PR #3740 - Fix includes in partitioned vector tests.
• PR #3739 - More fixes to make sure that std::flush really flushes all output.
• PR #3737 - Fix potential shutdown problems.
• PR #3736 - Fix guided_pool_executor after dataflow changes caused compilation fail.
• PR #3734 - Limiting executor.
• PR #3732 - More constrained bound constructors.
• PR #3730 - Attempt to fix deadlocks during component loading.
• PR #3729 - Add latch member function count_up and reset, requested by hpxMP.
• PR #3728 - Send even empty buffers on hpx::endl and hpx::flush.
• PR #372 - Adding example demonstrating how to customize the memory management for a component.
• PR #3726 - Adding support for passing command line options through the HPX_COMMANDLINE_OPTIONS environment variable.
• PR #3723 - Document known broken OpenMPI builds.
• PR #3716 - Add barrier reset function, requested by hpxMP for reusing barrier.
• PR #3715 - More work on functions and vtables.
• PR #3714 - Generate single-page HTML, PDF, manpage from documentation.
• PR #3713 - Updating default APEX version to 2.1.2.
• PR #3712 - Update release procedure.
• PR #3710 - Fix the C++11 build, after #3704.
• PR #3709 - Move some component_registry functionality to source file.

788 https://github.com/STEllAR-GROUP/hpx/pull/3748
789 https://github.com/STEllAR-GROUP/hpx/pull/3747
790 https://github.com/STEllAR-GROUP/hpx/pull/3745
791 https://github.com/STEllAR-GROUP/hpx/pull/3742
792 https://github.com/STEllAR-GROUP/hpx/pull/3740
793 https://github.com/STEllAR-GROUP/hpx/pull/3739
794 https://github.com/STEllAR-GROUP/hpx/pull/3737
795 https://github.com/STEllAR-GROUP/hpx/pull/3736
796 https://github.com/STEllAR-GROUP/hpx/pull/3734
797 https://github.com/STEllAR-GROUP/hpx/pull/3732
798 https://github.com/STEllAR-GROUP/hpx/pull/3730
799 https://github.com/STEllAR-GROUP/hpx/pull/3729
800 https://github.com/STEllAR-GROUP/hpx/pull/3728
801 https://github.com/STEllAR-GROUP/hpx/pull/3727
802 https://github.com/STEllAR-GROUP/hpx/pull/3726
803 https://github.com/STEllAR-GROUP/hpx/pull/3722
804 https://github.com/STEllAR-GROUP/hpx/pull/3716
805 https://github.com/STEllAR-GROUP/hpx/pull/3715
806 https://github.com/STEllAR-GROUP/hpx/pull/3714
807 https://github.com/STEllAR-GROUP/hpx/pull/3713
808 https://github.com/STEllAR-GROUP/hpx/pull/3712
809 https://github.com/STEllAR-GROUP/hpx/pull/3710
810 https://github.com/STEllAR-GROUP/hpx/pull/3709
• PR #3708 - Ignore all locks while handling assertions
• PR #3707 - Remove obsolete hpx runtime executable
• PR #3705 - Fix and simplify `make_ready_future` overload sets
• PR #3704 - Reduce use of binders
• PR #3703 - Ini
• PR #3702 - Fixing CUDA compiler errors
• PR #3700 - Added `barrier::increment` function to increase total number of thread
• PR #3699 - One more attempt to fix migration...
• PR #3694 - Fixing component migration
• PR #3693 - Print thread state when getting disallowed value in `set_thread_state`
• PR #3692 - Only disable `constexpr` with clang-cuda, not nvcc+gcc
• PR #3691 - Link with libsupc++ if needed for `thread_local`
• PR #3690 - Remove thousands separators in `set_operations_3442` to comply with C++11
• PR #3688 - Decouple serialization from function vtables
• PR #3687 - Fix a couple of test failures
• PR #3686 - Make sure tests.unit.build are run after install on CircleCI
• PR #3685 - Revise quickstart CMakeLists.txt explanation
• PR #3684 - Provide concept emulation for Ranges-TS concepts
• PR #3683 - Ignore uninitialized chunks
• PR #3682 - Ignore uninitialized chunks. Check proper indices.
• PR #3680 - Ignore uninitialized chunks. Check proper range indices
• PR #3679 - Simplify basic action implementations
• PR #3678 - Making sure `HPX_HAVE_LIBATOMIC` is unset before checking

2.11. Releases
• PR #3677 - Fix generated full version number to be usable in expressions
• PR #3674 - Reduce functional utilities call depth
• PR #3672 - Change new build system to use existing macros related to pseudo dependencies
• PR #3669 - Remove indirection in function_ref when thread description is disabled
• PR #3668 - Unbreaking async_*cb* tests
• PR #3667 - Generate version.hpp
• PR #3665 - Enabling MPI parcelport for gitlab runners
• PR #3664 - Making clang-tidy work properly again
• PR #3662 - Attempt to fix exception handling
• PR #3661 - Move lcos::latch to source file
• PR #3660 - Fix accidentally explicit gid_type default constructor
• PR #3659 - Parallel executor latch
• PR #3658 - Fixing execution_parameters
• PR #3657 - Avoid dangling references in wait_all
• PR #3656 - Avoiding lifetime problems with sync_put_parcel
• PR #3655 - Fixing nullptr dereference inside of function
• PR #3652 - Attempt to fix thread_map_type definition with C++11
• PR #3650 - Allowing for end iterator being different from begin iterator
• PR #3649 - Added architecture identification to cmake to be able to detect timestamp support
• PR #3648 - Enabling sanitizers on gitlab runner
• PR #3647 - Dataflow now works with functions that return a reference
• PR #3637 - Merging the executor-enabled overloads of `shared_future<>::then`
• PR #3633 - Replace deprecated boost endian macros
• PR #3639 - Add instructions on getting HPX to documentation
• PR #3631 - Simplify parcel creation
• PR #3630 - Small additions and fixes to release procedure
• PR #3629 - Modular pp
• PR #3627 - Implement `util::function_ref`
• PR #3626 - Fix cancelable_action_client example
• PR #3625 - Added automatic serialization for simple structs (see #3034)
• PR #3624 - Updating the default order of priority for `thread_description`
• PR #3621 - Update copyright year and other small formatting fixes
• PR #3620 - Adding support for gitlab runner
• PR #3619 - Store debug logs and core dumps on CircleCI
• PR #3618 - Various optimizations
• PR #3617 - Fix link to the gpg key (#2)
• PR #3615 - Fix unused variable warnings with networking off
• PR #3614 - Restructuring counter data in scheduler to reduce false sharing
• PR #3613 - Adding support for gitlab runners
• PR #3610 - Don’t wait for `stop_condition` in main thread
• PR #3608 - Add inline keyword to `invalid_thread_id` definition for nvcc
• PR #3607 - Adding configuration key that allows to explicitly add a directory to the component search path
• PR #3606 - Add nvcc to exclude constexpress since it is not supported by nvcc
• PR #3605 - Add inline to definition of checkpoint stream operators to fix link error
- PR #3604 - Use format for string formatting
- PR #3603 - Improve the error message for using to less MAX_CPU_COUNT
- PR #3602 - Improve the error message for to small values of MAX_CPU_COUNT
- PR #3600 - Parallel executor aggregated
- PR #3598 - Making sure networking is disabled for default one-locality-runs
- PR #3596 - Store thread exit functions in forward_list instead of deque to avoid allocations
- PR #3590 - Fix typo/mistake in thread queue cleanup_terminated
- PR #3588 - Fix formatting errors in launching_and_configuring_hpx_applications.rst
- PR #3586 - Make bind propagate value category
- PR #3585 - Extend Cmake for building hpx as distribution packages (refs #3575)
- PR #3584 - Untangle function storage from object pointer
- PR #3582 - Towards Modularized HPX
- PR #3580 - Remove extra | | in merge.hpp
- PR #3577 - Partially revert “Remove vtable empty flag”
- PR #3576 - Make sure empty startup/shutdown functions are not being used
- PR #3574 - Make sure DATAPAR settings are conveyed to depending projects
- PR #3573 - Make sure HPX is usable with latest released version of Vc (V1.4.1)
- PR #3572 - Adding test ensuring ticket 3565 is fixed
- PR #3571 - Make empty [unique_]function vtable non-dependent
- PR #3566 - Fix compilation with dynamic bitset for CPU masks
- PR #3563 - Drop util::[unique_]function target_type
- PR #3562 - Removing the target suffixes
- PR #3561 - Replace executor traits return type deduction (keep non-SFINAE)
• PR #3557 - Replace the last usages of boost::atomic
• PR #3556 - Replace boost::scoped_array with std::unique_ptr
• PR #3552 - (Re)move APEX readme
• PR #3548 - Replace boost::scoped_ptr with std::unique_ptr
• PR #3547 - Remove last use of Boost.Signals2
• PR #3544 - Post 1.2.0 version bumps
• PR #3543 - added Ubuntu dependency list to readme
• PR #3531 - Warnings, warnings...
• PR #3527 - Add CircleCI filter for building all tags
• PR #3525 - Segmented algorithms
• PR #3514 - Cleaning up the build system
• PR #3505 - Fixing type attribute warning for transfer_action
• PR #3504 - Add support for rpm packaging
• PR #3499 - Improving spinlock pools
• PR #3498 - Remove thread specific ptr
• PR #3486 - Fix comparison for expect_connecting_localities config entry
• PR #3469 - Enable (existing) code for extracting stack pointer on Power platform

2.11.4 HPX V1.2.1 (Feb 19, 2019)

General changes
This is a bugfix release. It contains the following changes:

• Fix compilation on ARM, s390x and 32-bit architectures.
• Fix a critical bug in the future implementation.
• Fix several problems in the CMake configuration which affects external projects.
• Add support for Boost 1.69.0.

---

903 https://github.com/STEllAR-GROUP/hpx/pull/3557
904 https://github.com/STEllAR-GROUP/hpx/pull/3556
905 https://github.com/STEllAR-GROUP/hpx/pull/3552
906 https://github.com/STEllAR-GROUP/hpx/pull/3548
907 https://github.com/STEllAR-GROUP/hpx/pull/3547
908 https://github.com/STEllAR-GROUP/hpx/pull/3544
909 https://github.com/STEllAR-GROUP/hpx/pull/3543
910 https://github.com/STEllAR-GROUP/hpx/pull/3531
911 https://github.com/STEllAR-GROUP/hpx/pull/3527
912 https://github.com/STEllAR-GROUP/hpx/pull/3525
913 https://github.com/STEllAR-GROUP/hpx/pull/3517
914 https://github.com/STEllAR-GROUP/hpx/pull/3514
915 https://github.com/STEllAR-GROUP/hpx/pull/3505
916 https://github.com/STEllAR-GROUP/hpx/pull/3504
917 https://github.com/STEllAR-GROUP/hpx/pull/3499
918 https://github.com/STEllAR-GROUP/hpx/pull/3498
919 https://github.com/STEllAR-GROUP/hpx/pull/3486
920 https://github.com/STEllAR-GROUP/hpx/pull/3469
Closed issues

- Issue #3638 - Build HPX 1.2 with boost 1.69
- Issue #3635 - Non-deterministic crashing on Stampede2
- Issue #3550 - 1>e:000workhpxsrcthrow_exception.cpp(54): error C2440: ‘<function-style-cast>’: cannot convert from ‘boost::system::error_code’ to ‘hpx::exception’
- Issue #3549 - HPX 1.2.0 does not build on i686, but release candidate did
- Issue #3511 - Build on s390x fails
- Issue #3509 - Build on armv7l fails

Closed pull requests

- PR #3695 - Don’t install CMake templates and packaging files
- PR #3666 - Fixing yet another race in future_data
- PR #3669 - Fixing race between setting and getting the value inside future_data
- PR #3648 - Adding timestamp option for S390x platform
- PR #3647 - Blind attempt to fix warnings issued by gcc V9
- PR #3611 - Include GNUInstallDirs earlier to have it available for subdirectories
- PR #3595 - Use GNUInstallDirs lib path in pkgconfig config file
- PR #3597 - Add include(GNUInstallDirs) to HPXMacros.cmake
- PR #3591 - Fix compilation error on arm7 architecture. Compiles and runs on Fedora 29 on Pi 3.
- PR #3558 - Adding constructor exception(boost::system::error_code const&)
- PR #3555 - cmake: make install locations configurable
- PR #3551 - Fix uint64_t causing compilation fail on i686

2.11.5 HPX V1.2.0 (Nov 12, 2018)

General changes

Here are some of the main highlights and changes for this release:

921 https://github.com/STEllAR-GROUP/hpx/issues/3638
922 https://github.com/STEllAR-GROUP/hpx/issues/3635
923 https://github.com/STEllAR-GROUP/hpx/issues/3550
924 https://github.com/STEllAR-GROUP/hpx/issues/3549
925 https://github.com/STEllAR-GROUP/hpx/issues/3511
926 https://github.com/STEllAR-GROUP/hpx/issues/3509
927 https://github.com/STEllAR-GROUP/hpx/pull/3695
928 https://github.com/STEllAR-GROUP/hpx/pull/3666
929 https://github.com/STEllAR-GROUP/hpx/pull/3663
930 https://github.com/STEllAR-GROUP/hpx/pull/3648
931 https://github.com/STEllAR-GROUP/hpx/pull/3647
932 https://github.com/STEllAR-GROUP/hpx/pull/3611
933 https://github.com/STEllAR-GROUP/hpx/pull/3595
934 https://github.com/STEllAR-GROUP/hpx/pull/3593
935 https://github.com/STEllAR-GROUP/hpx/pull/3591
936 https://github.com/STEllAR-GROUP/hpx/pull/3558
937 https://github.com/STEllAR-GROUP/hpx/pull/3555
938 https://github.com/STEllAR-GROUP/hpx/pull/3551
• Thanks to the work of our Google Summer of Code student, Nikunj Gupta, we now have a new implementation of `hpx_main.hpp` on supported platforms (Linux, BSD and MacOS). This is intended to be a less fragile drop-in replacement for the old implementation relying on preprocessor macros. The new implementation does not require changes if you are using the CMake or pkg-config. The old behaviour can be restored by setting `HPX_WITH_DYNAMIC_HPX_MAIN=OFF` during CMake configuration. The implementation on Windows is unchanged.

• We have added functionality to allow passing scheduling hints to our schedulers. These will allow us to create executors that for example target a specific NUMA domain or allow for HPX threads to be pinned to a particular worker thread.

• We have significantly improved the performance of our futures implementation by making the shared state atomic.

• We have replaced Boostbook by Sphinx for our documentation. This means the documentation is easier to navigate with built-in search and table of contents. We have also added a quick start section and restructured the documentation to be easier to follow for new users.

• We have added a new option to the `--hpx:threads` command line option. It is now possible to use `cores` to tell HPX to only use one worker thread per core, unlike the existing option `all` which uses one worker thread per processing unit (processing unit can be a hyperthread if hyperthreads are available). The default value of `--hpx:threads` has also been changed to `cores` as this leads to better performance in most cases.

• All command line options can now be passed alongside configuration options when initializing HPX. This means that some options that were previously only available on the command line can now be set as configuration options.

• HPXMP is a portable, scalable, and flexible application programming interface using the OpenMP specification that supports multi-platform shared memory multiprocessing programming in C and C++. HPXMP can be enabled within HPX by setting `DHPX_WITH_HPXMP=ON` during CMake configuration.

• Two new performance counters were added for measuring the time spent doing background work. `/threads/time/background-work-duration` returns the time spent doing background on a given thread or locality, while `/threads/time/background-overhead` returns the fraction of time spent doing background work with respect to the overall time spent running the scheduler. The new performance counters are disabled by default and can be turned on by setting `HPX_WITH_BACKGROUND_THREAD_COUNTERS=ON` during CMake configuration.

• The idling behaviour of HPX has been tweaked to allow for faster idling. This is useful in interactive applications where the HPX worker threads may not have work all the time. This behaviour can be tweaked and turned off as before with `HPX_WITH_THREAD_MANAGER_IDLE_BACKOFF=OFF` during CMake configuration.

• It is now possible to register callback functions for HPX worker thread events. Callbacks can be registered for starting and stopping worker threads, and for when errors occur.

### Breaking changes

• The implementation of `hpx_main.hpp` has changed. If you are using custom Makefiles you will need to make changes. Please see the documentation on using Makefiles for more details.

• The default value of `--hpx:threads` has changed from `all` to `cores`. The new option `cores` only starts one worker thread per core.

• We have dropped support for Boost 1.56 and 1.57. The minimal version of Boost we now test is 1.58.

---

939 https://www.cmake.org
940 https://www.cmake.org
941 https://www.cmake.org
942 https://www.cmake.org
943 https://www.cmake.org

2.11. Releases
• Our `boost::format`-based formatting implementation has been revised and replaced with a custom implementation. This changes the formatting syntax and requires changes if you are relying on `hpx::util::format` or `hpx::util::format_to`. The pull request for this change contains more information: PR #3266\(^{944}\).

• The following deprecated options have now been completely removed:
  - `HPX_WITH_ASYNC_FUNCTION_COMPATIBILITY`
  - `HPX_WITH_LOCAL_DATAFLOW`
  - `HPX_WITH_GENERIC_EXECUTION_POLICY`
  - `HPX_WITH_BOOST_CHRONO_COMPATIBILITY`
  - `HPX_WITH_EXECUTOR_COMPATIBILITY`
  - `HPX_WITH_EXECUTION_POLICY_COMPATIBILITY`
  - `HPX_WITH_TRANSFORM_REDUCE_COMPATIBILITY`.

Closed issues

  - Issue #3538\(^{945}\) - numa handling incorrect for hwloc 2
  - Issue #3533\(^{946}\) - Cmake version 3.5.1 does not work (git ff26b35 2018-11-06)
  - Issue #3526\(^{947}\) - Failed building hpx-1.2.0-rc1 on Ubuntu16.04 x86-64 Virtualbox VM
  - Issue #3512\(^{948}\) - Build on aarch64 fails
  - Issue #3475\(^{949}\) - HPX fails to link if the MPI parcelport is enabled
  - Issue #3462\(^{950}\) - CMake configuration shows a minor and inconsequential failure to create a symlink
  - Issue #3461\(^{951}\) - Compilation Problems with the most recent Clang
  - Issue #3460\(^{952}\) - Deadlock when create_partitioner fails (assertion fails) in debug mode
  - Issue #3455\(^{953}\) - HPX build failing with HWLOC errors on POWER8 with hwloc 1.8
  - Issue #3438\(^{954}\) - HPX no longer builds on IBM POWER8
  - Issue #3426\(^{955}\) - hpx build failed on MacOS
  - Issue #3424\(^{956}\) - CircleCI builds broken for forked repositories
  - Issue #3422\(^{957}\) - Benchmarks in tests.performance.local are not run nightly
  - Issue #3408\(^{958}\) - CMake Targets for HPX
  - Issue #3399\(^{959}\) - processing unit out of bounds
  - Issue #3395\(^{960}\) - Floating point bug in hpx/runtime/threads/policies/scheduler_base.hpp
  - Issue #3378\(^{961}\) - compile error with lcos::communicator

---

\(^{944}\) https://github.com/STEllAR-GROUP/hpx/pull/3266
\(^{945}\) https://github.com/STEllAR-GROUP/hpx/issues/3538
\(^{946}\) https://github.com/STEllAR-GROUP/hpx/issues/3533
\(^{947}\) https://github.com/STEllAR-GROUP/hpx/issues/3526
\(^{948}\) https://github.com/STEllAR-GROUP/hpx/issues/3512
\(^{949}\) https://github.com/STEllAR-GROUP/hpx/issues/3475
\(^{950}\) https://github.com/STEllAR-GROUP/hpx/issues/3462
\(^{951}\) https://github.com/STEllAR-GROUP/hpx/issues/3461
\(^{952}\) https://github.com/STEllAR-GROUP/hpx/issues/3460
\(^{953}\) https://github.com/STEllAR-GROUP/hpx/issues/3455
\(^{954}\) https://github.com/STEllAR-GROUP/hpx/issues/3438
\(^{955}\) https://github.com/STEllAR-GROUP/hpx/issues/3426
\(^{956}\) https://github.com/STEllAR-GROUP/hpx/issues/3424
\(^{957}\) https://github.com/STEllAR-GROUP/hpx/issues/3422
\(^{958}\) https://github.com/STEllAR-GROUP/hpx/issues/3408
\(^{959}\) https://github.com/STEllAR-GROUP/hpx/issues/3399
\(^{960}\) https://github.com/STEllAR-GROUP/hpx/issues/3395
\(^{961}\) https://github.com/STEllAR-GROUP/hpx/issues/3378
• Issue #3376 - Failed to build HPX with APEX using clang
• Issue #3366 - Adapted Safe_Object example fails for -hpx:threads > 1
• Issue #3360 - Segmentation fault when passing component id as parameter
• Issue #3358 - HPX runtime hangs after multiple (~thousands) start-stop sequences
• Issue #3352 - Support TCP provider in libfabric ParcelPort
• Issue #3342 - undefined reference to __atomic_load_16
• Issue #3339 - setting command line options/flags from init cfg is not obvious
• Issue #3325 - AGAS migrates components prematurely
• Issue #3321 - hpx bad parameter handling is awful
• Issue #3318 - Benchmarks fail to build with C++11
• Issue #3304 - hpx::threads::run_as_hpx_thread does not properly handle exceptions
• Issue #3300 - Setting pu step or offset results in no threads in default pool
• Issue #3297 - Crash with APEX when running Phylanx lra_csv with > 1 thread
• Issue #3296 - Building HPX with APEX configuration gives compiler warnings
• Issue #3290 - make tests failing at hello_world_component
• Issue #3285 - possible compilation error when “using namespace std;” is defined before including “hpx” headers files
• Issue #3280 - HPX fails on OSX
• Issue #3272 - CircleCI does not upload generated docker image any more
• Issue #3270 - Error when compiling CUDA examples
• Issue #3267 - tests.unit.host_.block_allocator fails occasionally
• Issue #3264 - Possible move to Sphinx for documentation
• Issue #3263 - Documentation improvements
• Issue #3259 - set_parcel_write_handler test fails occasionally

962 https://github.com/STEllAR-GROUP/hpx/issues/3376
963 https://github.com/STEllAR-GROUP/hpx/issues/3366
964 https://github.com/STEllAR-GROUP/hpx/issues/3360
965 https://github.com/STEllAR-GROUP/hpx/issues/3358
966 https://github.com/STEllAR-GROUP/hpx/issues/3352
967 https://github.com/STEllAR-GROUP/hpx/issues/3342
968 https://github.com/STEllAR-GROUP/hpx/issues/3339
969 https://github.com/STEllAR-GROUP/hpx/issues/3325
970 https://github.com/STEllAR-GROUP/hpx/issues/3321
971 https://github.com/STEllAR-GROUP/hpx/issues/3318
972 https://github.com/STEllAR-GROUP/hpx/issues/3304
973 https://github.com/STEllAR-GROUP/hpx/issues/3300
974 https://github.com/STEllAR-GROUP/hpx/issues/3297
975 https://github.com/STEllAR-GROUP/hpx/issues/3296
976 https://github.com/STEllAR-GROUP/hpx/issues/3290
977 https://github.com/STEllAR-GROUP/hpx/issues/3285
978 https://github.com/STEllAR-GROUP/hpx/issues/3280
979 https://github.com/STEllAR-GROUP/hpx/issues/3272
980 https://github.com/STEllAR-GROUP/hpx/issues/3270
981 https://github.com/STEllAR-GROUP/hpx/issues/3267
982 https://github.com/STEllAR-GROUP/hpx/issues/3264
983 https://github.com/STEllAR-GROUP/hpx/issues/3263
984 https://github.com/STEllAR-GROUP/hpx/issues/3259
• Issue #3258<sup>985</sup> - Links to source code in documentation are broken
• Issue #3247<sup>986</sup> - Rare tests.unit.host_.block_allocator test failure on 1.1.0-rc1
• Issue #3244<sup>987</sup> - Slowing down and speeding up an interval_timer
• Issue #3215<sup>988</sup> - Cannot build both tests and examples on MSVC with pseudo-dependencies enabled
• Issue #3195<sup>989</sup> - Unnecessary customization point route causing performance penalty
• Issue #3088<sup>990</sup> - A strange thing in parallel::sort.
• Issue #2650<sup>991</sup> - libfabric support for passive endpoints
• Issue #1205<sup>992</sup> - TSS is broken

Closed pull requests

• PR #3542<sup>993</sup> - Fix numa lookup from pu when using hwloc 2.x
• PR #3541<sup>994</sup> - Fixing the build system of the MPI parcelport
• PR #3540<sup>995</sup> - Updating HPX people section
• PR #3539<sup>996</sup> - Splitting test to avoid OOM on CircleCI
• PR #3537<sup>997</sup> - Fix guided exec
• PR #3536<sup>998</sup> - Updating grants which support the LSU team
• PR #3535<sup>999</sup> - Fix hiding of docker credentials
• PR #3534<sup>1000</sup> - Fixing #3533
• PR #3532<sup>1001</sup> - fixing minor doc typo --hpx:print-counter-at arg
• PR #3531<sup>1002</sup> - Changing APEX default tag to v2.1.0
• PR #3529<sup>1003</sup> - Remove leftover security options and documentation
• PR #3528<sup>1004</sup> - Fix hwloc version check
• PR #3524<sup>1005</sup> - Do not build guided pool examples with older GCC compilers
• PR #3523<sup>1006</sup> - Fix logging regression

<sup>985</sup> https://github.com/STEllAR-GROUP/hpx/issues/3258
<sup>986</sup> https://github.com/STEllAR-GROUP/hpx/issues/3247
<sup>987</sup> https://github.com/STEllAR-GROUP/hpx/issues/3244
<sup>988</sup> https://github.com/STEllAR-GROUP/hpx/issues/3215
<sup>989</sup> https://github.com/STEllAR-GROUP/hpx/issues/3195
<sup>990</sup> https://github.com/STEllAR-GROUP/hpx/issues/3088
<sup>991</sup> https://github.com/STEllAR-GROUP/hpx/issues/2650
<sup>992</sup> https://github.com/STEllAR-GROUP/hpx/issues/1205
<sup>993</sup> https://github.com/STEllAR-GROUP/hpx/pull/3542
<sup>994</sup> https://github.com/STEllAR-GROUP/hpx/pull/3541
<sup>995</sup> https://github.com/STEllAR-GROUP/hpx/pull/3540
<sup>996</sup> https://github.com/STEllAR-GROUP/hpx/pull/3539
<sup>997</sup> https://github.com/STEllAR-GROUP/hpx/pull/3537
<sup>998</sup> https://github.com/STEllAR-GROUP/hpx/pull/3536
<sup>999</sup> https://github.com/STEllAR-GROUP/hpx/pull/3535
<sup>1000</sup> https://github.com/STEllAR-GROUP/hpx/pull/3534
<sup>1001</sup> https://github.com/STEllAR-GROUP/hpx/pull/3532
<sup>1002</sup> https://github.com/STEllAR-GROUP/hpx/pull/3530
<sup>1003</sup> https://github.com/STEllAR-GROUP/hpx/pull/3529
<sup>1004</sup> https://github.com/STEllAR-GROUP/hpx/pull/3528
<sup>1005</sup> https://github.com/STEllAR-GROUP/hpx/pull/3524
<sup>1006</sup> https://github.com/STEllAR-GROUP/hpx/pull/3523
- PR #3522\textsuperscript{1007} - Fix more warnings
- PR #3521\textsuperscript{1008} - Fixing argument handling in induction and reduction clauses for parallel::for_loop
- PR #3520\textsuperscript{1009} - Remove docs symlink and versioned docs folders
- PR #3519\textsuperscript{1010} - hpxMP release
- PR #3518\textsuperscript{1011} - Change all steps to use new docker image on CircleCI
- PR #3516\textsuperscript{1012} - Drop usage of deprecated facilities removed in C++17
- PR #3515\textsuperscript{1013} - Remove remaining uses of Boost.TypeTraits
- PR #3513\textsuperscript{1014} - Fixing a CMake problem when trying to use libfabric
- PR #3508\textsuperscript{1015} - Remove memory_block component
- PR #3507\textsuperscript{1016} - Propagating the MPI compile definitions to all relevant targets
- PR #3503\textsuperscript{1017} - Update documentation colors and logo
- PR #3502\textsuperscript{1018} - Fix bogus ‘throws’ bindings in scheduled_thread_pool_impl
- PR #3501\textsuperscript{1019} - Split parallel::remove_if tests to avoid OOM on CircleCI
- PR #3500\textsuperscript{1020} - Support NONAMEPREFIX in add_hpx_library()
- PR #3497\textsuperscript{1021} - Note that cuda support requires cmake 3.9
- PR #3495\textsuperscript{1022} - Fixing dataflow
- PR #3493\textsuperscript{1023} - Remove deprecated options for 1.2.0 part 2
- PR #3492\textsuperscript{1024} - Add CUDA_LINK_LIBRARIES_KEYWOD to allow PRIVATE keyword in linkage t...
- PR #3491\textsuperscript{1025} - Changing Base docker image
- PR #3490\textsuperscript{1026} - Don’t create tasks immediately with hpx::apply
- PR #3489\textsuperscript{1027} - Remove deprecated options for 1.2.0
- PR #3488\textsuperscript{1028} - Revert “Use BUILD_INTERFACE generator expression to fix cmake flag exports”
- PR #3487\textsuperscript{1029} - Revert “Fixing type attribute warning for transfer_action”

\textsuperscript{1007}https://github.com/STEllAR-GROUP/hpx/pull/3522
\textsuperscript{1008}https://github.com/STEllAR-GROUP/hpx/pull/3521
\textsuperscript{1009}https://github.com/STEllAR-GROUP/hpx/pull/3520
\textsuperscript{1010}https://github.com/STEllAR-GROUP/hpx/pull/3519
\textsuperscript{1011}https://github.com/STEllAR-GROUP/hpx/pull/3518
\textsuperscript{1012}https://github.com/STEllAR-GROUP/hpx/pull/3516
\textsuperscript{1013}https://github.com/STEllAR-GROUP/hpx/pull/3515
\textsuperscript{1014}https://github.com/STEllAR-GROUP/hpx/pull/3513
\textsuperscript{1015}https://github.com/STEllAR-GROUP/hpx/pull/3508
\textsuperscript{1016}https://github.com/STEllAR-GROUP/hpx/pull/3507
\textsuperscript{1017}https://github.com/STEllAR-GROUP/hpx/pull/3503
\textsuperscript{1018}https://github.com/STEllAR-GROUP/hpx/pull/3502
\textsuperscript{1019}https://github.com/STEllAR-GROUP/hpx/pull/3501
\textsuperscript{1020}https://github.com/STEllAR-GROUP/hpx/pull/3500
\textsuperscript{1021}https://github.com/STEllAR-GROUP/hpx/pull/3497
\textsuperscript{1022}https://github.com/STEllAR-GROUP/hpx/pull/3495
\textsuperscript{1023}https://github.com/STEllAR-GROUP/hpx/pull/3493
\textsuperscript{1024}https://github.com/STEllAR-GROUP/hpx/pull/3492
\textsuperscript{1025}https://github.com/STEllAR-GROUP/hpx/pull/3491
\textsuperscript{1026}https://github.com/STEllAR-GROUP/hpx/pull/3490
\textsuperscript{1027}https://github.com/STEllAR-GROUP/hpx/pull/3489
\textsuperscript{1028}https://github.com/STEllAR-GROUP/hpx/pull/3488
\textsuperscript{1029}https://github.com/STEllAR-GROUP/hpx/pull/3487
• PR #3485 - Use BUILD_INTERFACE generator expression to fix cmake flag exports
• PR #3483 - Fixing type attribute warning for transfer_action
• PR #3481 - Remove unused variables
• PR #3480 - Towards a more lightweight transfer action
• PR #3479 - Fix FLGS - Use correct version of target_compile_options
• PR #3478 - Making sure the application’s exit code is properly propagated back to the OS
• PR #3476 - Don’t print docker credentials as part of the environment.
• PR #3473 - Fixing invalid cmake code if no jemalloc prefix was given
• PR #3472 - Attempting to work around recent clang test compilation failures
• PR #3471 - Enable jemalloc on windows
• PR #3470 - Updates readme
• PR #3469 - Avoid hang if there is an exception thrown during startup
• PR #3467 - Add compiler specific fallthrough attributes if C++17 attribute is not available
• PR #3466 - - bugfix : fix compilation with llvm-7.0
• PR #3465 - This patch adds various optimizations extracted from the thread_local_allocator work
• PR #3464 - Check for forked repos in CircleCI docker push step
• PR #3463 - - cmake : create the parent directory before symlinking
• PR #3459 - Remove unused/incomplete functionality from util/logging
• PR #3458 - Fix a problem with scope of CMAKE_CXX_FLAGS and hpx_add_compile_flag
• PR #3457 - Fixing more size_t -> int16_t (and similar) warnings
• PR #3456 - Add ifndefs to topology.cpp to support old hwloc versions again
• PR #3454 - Fixing warnings related to silent conversion of size_t -> int16_t
• PR #3451 - Add examples as unit tests

1030 https://github.com/STEllAR-GROUP/hpx/pull/3485
1031 https://github.com/STEllAR-GROUP/hpx/pull/3483
1032 https://github.com/STEllAR-GROUP/hpx/pull/3481
1033 https://github.com/STEllAR-GROUP/hpx/pull/3480
1034 https://github.com/STEllAR-GROUP/hpx/pull/3479
1035 https://github.com/STEllAR-GROUP/hpx/pull/3478
1036 https://github.com/STEllAR-GROUP/hpx/pull/3476
1037 https://github.com/STEllAR-GROUP/hpx/pull/3473
1038 https://github.com/STEllAR-GROUP/hpx/pull/3472
1039 https://github.com/STEllAR-GROUP/hpx/pull/3471
1040 https://github.com/STEllAR-GROUP/hpx/pull/3470
1041 https://github.com/STEllAR-GROUP/hpx/pull/3468
1042 https://github.com/STEllAR-GROUP/hpx/pull/3467
1043 https://github.com/STEllAR-GROUP/hpx/pull/3466
1044 https://github.com/STEllAR-GROUP/hpx/pull/3465
1045 https://github.com/STEllAR-GROUP/hpx/pull/3464
1046 https://github.com/STEllAR-GROUP/hpx/pull/3463
1047 https://github.com/STEllAR-GROUP/hpx/pull/3459
1048 https://github.com/STEllAR-GROUP/hpx/pull/3458
1049 https://github.com/STEllAR-GROUP/hpx/pull/3457
1050 https://github.com/STEllAR-GROUP/hpx/pull/3456
1051 https://github.com/STEllAR-GROUP/hpx/pull/3454
1052 https://github.com/STEllAR-GROUP/hpx/pull/3451
• PR #3450 - Constexpr-fying bind and other functional facilities
• PR #3446 - Fix some thread suspension timeouts
• PR #3445 - Fix various warnings
• PR #3443 - Only enable service pool config options if pools are enabled
• PR #3441 - Fix missing closing brackets in documentation
• PR #3439 - Use correct MPI CXX libraries for MPI parcelport
• PR #3436 - Add projection function to find_* (and fix very bad bug)
• PR #3435 - Fixing 1205
• PR #3434 - Fix threads cores
• PR #3433 - Add Heise Online to release announcement list
• PR #3432 - Don’t track task dependencies for distributed runs
• PR #3431 - Circle CI setting changes for hpxMP
• PR #3430 - Fix unused params warning
• PR #3429 - One thread per core
• PR #3428 - This suppresses a deprecation warning that is being issued by MSVC 19.15.26726
• PR #3427 - Fixes #3426
• PR #3425 - Use source cache and workspace between job steps on CircleCI
• PR #3421 - Add CDash timing output to future overhead test (for graphs)
• PR #3420 - Add guided_pool_executor
• PR #3419 - Fix typo in CircleCI config
• PR #3418 - Add sphinx documentation
• PR #3415 - Scheduler NUMA hint and shared priority scheduler
• PR #3414 - Adding step to synchronize the APEX release

1053 https://github.com/STEllAR-GROUP/hpx/pull/3450
1054 https://github.com/STEllAR-GROUP/hpx/pull/3446
1055 https://github.com/STEllAR-GROUP/hpx/pull/3445
1056 https://github.com/STEllAR-GROUP/hpx/pull/3443
1057 https://github.com/STEllAR-GROUP/hpx/pull/3441
1058 https://github.com/STEllAR-GROUP/hpx/pull/3439
1059 https://github.com/STEllAR-GROUP/hpx/pull/3436
1060 https://github.com/STEllAR-GROUP/hpx/pull/3435
1061 https://github.com/STEllAR-GROUP/hpx/pull/3434
1062 https://github.com/STEllAR-GROUP/hpx/pull/3433
1063 https://github.com/STEllAR-GROUP/hpx/pull/3432
1064 https://github.com/STEllAR-GROUP/hpx/pull/3431
1065 https://github.com/STEllAR-GROUP/hpx/pull/3430
1066 https://github.com/STEllAR-GROUP/hpx/pull/3429
1067 https://github.com/STEllAR-GROUP/hpx/pull/3428
1068 https://github.com/STEllAR-GROUP/hpx/pull/3427
1069 https://github.com/STEllAR-GROUP/hpx/pull/3425
1070 https://github.com/STEllAR-GROUP/hpx/pull/3421
1071 https://github.com/STEllAR-GROUP/hpx/pull/3420
1072 https://github.com/STEllAR-GROUP/hpx/pull/3419
1073 https://github.com/STEllAR-GROUP/hpx/pull/3418
1074 https://github.com/STEllAR-GROUP/hpx/pull/3415
1075 https://github.com/STEllAR-GROUP/hpx/pull/3414

2.11. Releases
• PR #3413\textsuperscript{1076} - Fixing multiple defines of APEX_HAVE_HPX

• PR #3412\textsuperscript{1077} - Fixes linking with libhpx\_wrap error with BSD and Windows based systems

• PR #3410\textsuperscript{1078} - Fix typo in CMakeLists.txt

• PR #3409\textsuperscript{1079} - Fix brackets and indentation in existing\_performance\_counters.qbk

• PR #3407\textsuperscript{1080} - Fix unused param and extra ; warnings emitted by gcc 8.x

• PR #3406\textsuperscript{1081} - Adding thread local allocator and use it for future shared states

• PR #3405\textsuperscript{1082} - Adding DHPX\_HAVE\_THREAD\_LOCAL\_STORAGE=ON to builds

• PR #3404\textsuperscript{1083} - fixing multiple definition of main() in linux

• PR #3402\textsuperscript{1084} - Allow debug option to be enabled only for Linux systems with dynamic main on linux

• PR #3401\textsuperscript{1085} - Fix cuda\_future\_helper.h when compiling with C++11

• PR #3400\textsuperscript{1086} - Fix floating point exception scheduler\_base idle backoff

• PR #3398\textsuperscript{1087} - Atomic future state

• PR #3397\textsuperscript{1088} - Fixing code for older gcc versions

• PR #3396\textsuperscript{1089} - Allowing to register thread event functions (start/stop/error)

• PR #3394\textsuperscript{1090} - Fix small mistake in primary\_namespace\_server.cpp

• PR #3393\textsuperscript{1091} - Explicitly instantiate configured schedulers

• PR #3392\textsuperscript{1092} - Add performance counters background overhead and background work duration

• PR #3391\textsuperscript{1093} - Adapt integration of HPXMP to latest build system changes

• PR #3390\textsuperscript{1094} - Make AGAS measurements optional

• PR #3389\textsuperscript{1095} - Fix deadlock during shutdown

• PR #3388\textsuperscript{1096} - Add several functionalities allowing to optimize synchronous action invocation

• PR #3387\textsuperscript{1097} - Add cmake option to opt out of fail-compile tests

• PR #3386\textsuperscript{1098} - Adding support for boost::container::small\_vector to dataflow

\textsuperscript{1076}https://github.com/STEllAR-GROUP/hpx/pull/3413
\textsuperscript{1077}https://github.com/STEllAR-GROUP/hpx/pull/3412
\textsuperscript{1078}https://github.com/STEllAR-GROUP/hpx/pull/3410
\textsuperscript{1079}https://github.com/STEllAR-GROUP/hpx/pull/3409
\textsuperscript{1080}https://github.com/STEllAR-GROUP/hpx/pull/3407
\textsuperscript{1081}https://github.com/STEllAR-GROUP/hpx/pull/3406
\textsuperscript{1082}https://github.com/STEllAR-GROUP/hpx/pull/3405
\textsuperscript{1083}https://github.com/STEllAR-GROUP/hpx/pull/3404
\textsuperscript{1084}https://github.com/STEllAR-GROUP/hpx/pull/3402
\textsuperscript{1085}https://github.com/STEllAR-GROUP/hpx/pull/3401
\textsuperscript{1086}https://github.com/STEllAR-GROUP/hpx/pull/3400
\textsuperscript{1087}https://github.com/STEllAR-GROUP/hpx/pull/3398
\textsuperscript{1088}https://github.com/STEllAR-GROUP/hpx/pull/3397
\textsuperscript{1089}https://github.com/STEllAR-GROUP/hpx/pull/3396
\textsuperscript{1090}https://github.com/STEllAR-GROUP/hpx/pull/3394
\textsuperscript{1091}https://github.com/STEllAR-GROUP/hpx/pull/3393
\textsuperscript{1092}https://github.com/STEllAR-GROUP/hpx/pull/3392
\textsuperscript{1093}https://github.com/STEllAR-GROUP/hpx/pull/3391
\textsuperscript{1094}https://github.com/STEllAR-GROUP/hpx/pull/3390
\textsuperscript{1095}https://github.com/STEllAR-GROUP/hpx/pull/3389
\textsuperscript{1096}https://github.com/STEllAR-GROUP/hpx/pull/3388
\textsuperscript{1097}https://github.com/STEllAR-GROUP/hpx/pull/3387
\textsuperscript{1098}https://github.com/STEllAR-GROUP/hpx/pull/3386

908 Chapter 2. What’s so special about HPX?
• PR #3385 - Adds Debug option for hpx initializing from main
• PR #3384 - This hopefully fixes two tests that occasionally fail
• PR #3383 - Making sure thread local storage is enable for hpxMP
• PR #3382 - Fix usage of HPX_CAPTURE together with default value capture [=]
• PR #3381 - Replace undefined instantiations of uniform_int_distribution
• PR #3380 - Add missing semicolons to uses of HPX_COMPILER_fence
• PR #3379 - Fixing #3378
• PR #3377 - Adding build system support to integrate hpxmp into hpx at the user's machine
• PR #3375 - Replacing wrapper for __libc_start_main with main
• PR #3374 - Adds hpx_wrap to HPX_LINK_LIBRARIES which links only when specified.
• PR #3373 - Forcing cache settings in HPXConfig.cmake to guarantee updated values
• PR #3372 - Fixing some more c++11 build problems
• PR #3371 - Adds HPX_LINKER_FLAGS to HPX applications without editing their source codes
• PR #3370 - util::format: add typeSpecifier<> specializations for %!s(MISSING) and %!l(MISSING)s
• PR #3369 - Adding configuration option to allow explicit disable of the new hpx_main feature on Linux
• PR #3368 - Updates doc with recent hpx_wrap implementation
• PR #3367 - Adds Mac OS implementation to hpx_main.hpp
• PR #3366 - Fix order of hpx libs in HPX_CONF_LIBRARIES.
• PR #3365 - Apex fixing null wrapper
• PR #3364 - Making sure all parcels get destroyed on an HPX thread (TCP pp)
• PR #3359 - Feature/improveerrorforcompiler
• PR #3357 - Static/dynamic executable implementation
• PR #3355 - Reverting changes introduced by #3283 as those make applications hang

1099 https://github.com/STEllAR-GROUP/hpx/pull/3385
1100 https://github.com/STEllAR-GROUP/hpx/pull/3384
1101 https://github.com/STEllAR-GROUP/hpx/pull/3383
1102 https://github.com/STEllAR-GROUP/hpx/pull/3382
1103 https://github.com/STEllAR-GROUP/hpx/pull/3381
1104 https://github.com/STEllAR-GROUP/hpx/pull/3380
1105 https://github.com/STEllAR-GROUP/hpx/pull/3379
1106 https://github.com/STEllAR-GROUP/hpx/pull/3377
1107 https://github.com/STEllAR-GROUP/hpx/pull/3375
1108 https://github.com/STEllAR-GROUP/hpx/pull/3374
1109 https://github.com/STEllAR-GROUP/hpx/pull/3373
1110 https://github.com/STEllAR-GROUP/hpx/pull/3372
1111 https://github.com/STEllAR-GROUP/hpx/pull/3371
1112 https://github.com/STEllAR-GROUP/hpx/pull/3370
1113 https://github.com/STEllAR-GROUP/hpx/pull/3369
1114 https://github.com/STEllAR-GROUP/hpx/pull/3368
1115 https://github.com/STEllAR-GROUP/hpx/pull/3367
1116 https://github.com/STEllAR-GROUP/hpx/pull/3365
1117 https://github.com/STEllAR-GROUP/hpx/pull/3363
1118 https://github.com/STEllAR-GROUP/hpx/pull/3361
1119 https://github.com/STEllAR-GROUP/hpx/pull/3359
1120 https://github.com/STEllAR-GROUP/hpx/pull/3357
1121 https://github.com/STEllAR-GROUP/hpx/pull/3355
• PR #3354 - Add external dependencies to HPX_LIBRARY_DIR
• PR #3353 - Fix libfabric tcp
• PR #3351 - Move obsolete header to tests directory.
• PR #3350 - Renaming two functions to avoid problem described in #3285
• PR #3349 - Make idle backoff exponential with maximum sleep time
• PR #3347 - Replace simple_component* with component* in the Documentation
• PR #3346 - Fix CMakeLists.txt example in quick start
• PR #3345 - Fix automatic setting of HPX_MORE_THAN_64_THREADS
• PR #3344 - Reduce amount of information printed for unknown command line options
• PR #3343 - Safeguard HPX against destruction in global contexts
• PR #3341 - Allowing for all command line options to be used as configuration settings
• PR #3340 - Always convert inspect results to JUnit XML
• PR #3336 - Only run docker push on master on CircleCI
• PR #3335 - Update description of hpx.os_threads config parameter.
• PR #3334 - Making sure early logging settings don’t get mixed with others
• PR #3333 - Update CMake links and versions in documentation
• PR #3332 - Add notes on target suffixes to CMake documentation
• PR #3331 - Add quickstart section to documentation
• PR #3330 - Rename resource_partitioner test to avoid conflicts with pseudodependencies
• PR #3328 - Making sure object is pinned while executing actions, even if action returns a future
• PR #3327 - Add missing std::forward to tuple.hpp
• PR #3326 - Make sure logging is up and running while modules are being discovered.
• PR #3324 - Replace C++14 overload of std::equal with C++11 code.

https://github.com/STEllAR-GROUP/hpx/pull/3354
https://github.com/STEllAR-GROUP/hpx/pull/3353
https://github.com/STEllAR-GROUP/hpx/pull/3350
https://github.com/STEllAR-GROUP/hpx/pull/3349
https://github.com/STEllAR-GROUP/hpx/pull/3347
https://github.com/STEllAR-GROUP/hpx/pull/3346
https://github.com/STEllAR-GROUP/hpx/pull/3345
https://github.com/STEllAR-GROUP/hpx/pull/3344
https://github.com/STEllAR-GROUP/hpx/pull/3343
https://github.com/STEllAR-GROUP/hpx/pull/3341
https://github.com/STEllAR-GROUP/hpx/pull/3340
https://github.com/STEllAR-GROUP/hpx/pull/3336
https://github.com/STEllAR-GROUP/hpx/pull/3335
https://github.com/STEllAR-GROUP/hpx/pull/3334
https://github.com/STEllAR-GROUP/hpx/pull/3333
https://github.com/STEllAR-GROUP/hpx/pull/3332
https://github.com/STEllAR-GROUP/hpx/pull/3331
https://github.com/STEllAR-GROUP/hpx/pull/3330
https://github.com/STEllAR-GROUP/hpx/pull/3328
https://github.com/STEllAR-GROUP/hpx/pull/3327
https://github.com/STEllAR-GROUP/hpx/pull/3326
https://github.com/STEllAR-GROUP/hpx/pull/3324
- PR #3323\footnote{https://github.com/STEllAR-GROUP/hpx/pull/3323} - Fix a missing apex thread data (wrapper) initialization
- PR #3320\footnote{https://github.com/STEllAR-GROUP/hpx/pull/3320} - Adding support for -std=c++2a (define \texttt{HPX\_WITH\_CXX2A=On})
- PR #3319\footnote{https://github.com/STEllAR-GROUP/hpx/pull/3319} - Replacing C++14 feature with equivalent C++11 code
- PR #3317\footnote{https://github.com/STEllAR-GROUP/hpx/pull/3317} - Fix compilation with VS 15.7.1 and /std:c++latest
- PR #3316\footnote{https://github.com/STEllAR-GROUP/hpx/pull/3316} - Fix includes for 1d_stencil_*_omp examples
- PR #3314\footnote{https://github.com/STEllAR-GROUP/hpx/pull/3314} - Remove some unused parameter warnings
- PR #3313\footnote{https://github.com/STEllAR-GROUP/hpx/pull/3313} - Fix pu-step and pu-offset command line options
- PR #3312\footnote{https://github.com/STEllAR-GROUP/hpx/pull/3312} - Add conversion of inspect reports to JUnit XML
- PR #3311\footnote{https://github.com/STEllAR-GROUP/hpx/pull/3311} - Fix escaping of closing braces in format specification syntax
- PR #3310\footnote{https://github.com/STEllAR-GROUP/hpx/pull/3310} - Don’t overwrite user settings with defaults in registration database
- PR #3309\footnote{https://github.com/STEllAR-GROUP/hpx/pull/3309} - Fixing potential stack overflow for dataflow
- PR #3308\footnote{https://github.com/STEllAR-GROUP/hpx/pull/3308} - This updates the .clang-format configuration file to utilize newer features
- PR #3306\footnote{https://github.com/STEllAR-GROUP/hpx/pull/3306} - Marking migratable objects in their gid to allow not handling migration in AGAS
- PR #3305\footnote{https://github.com/STEllAR-GROUP/hpx/pull/3305} - Add proper exception handling to run\_as\_hpx\_thread
- PR #3303\footnote{https://github.com/STEllAR-GROUP/hpx/pull/3303} - Changed std::rand to a better inbuilt PRNG Generator
- PR #3302\footnote{https://github.com/STEllAR-GROUP/hpx/pull/3302} - All non-migratable (simple) components now encode their lva and component type in their gid
- PR #3301\footnote{https://github.com/STEllAR-GROUP/hpx/pull/3301} - Add nullptr\_t overloads to resource partitioner
- PR #3298\footnote{https://github.com/STEllAR-GROUP/hpx/pull/3298} - Apex task wrapper memory bug
- PR #3295\footnote{https://github.com/STEllAR-GROUP/hpx/pull/3295} - Fix mistakes after merge of CircleCI config
- PR #3294\footnote{https://github.com/STEllAR-GROUP/hpx/pull/3294} - Fix partitioned vector include in partitioned\_vector\_find tests
- PR #3293\footnote{https://github.com/STEllAR-GROUP/hpx/pull/3293} - Adding emplace support to promise and make\_ready\_future
- PR #3292\footnote{https://github.com/STEllAR-GROUP/hpx/pull/3292} - Add new cuda kernel synchronization with hpx\:\:future demo
- PR #3291\footnote{https://github.com/STEllAR-GROUP/hpx/pull/3291} - Fixes #3290
• PR #3289 - Fixing Docker image creation
• PR #3288 - Avoid allocating shared state for wait_all
• PR #3287 - Fixing /scheduler/utilization/instantaneous performance counter
• PR #3286 - dataflow() and future::then() use sync policy where possible
• PR #3284 - Background thread can use relaxed atomics to manipulate thread state
• PR #3283 - Do not unwrap ready future
• PR #3282 - Fix virtual method override warnings in static schedulers
• PR #3281 - Disable set_area_membind_nodeset for OSX
• PR #3279 - Add two variations to the future_overhead benchmark
• PR #3278 - Fix circleci workspace
• PR #3277 - Support external plugins
• PR #3276 - Fix missing parenthesis in hello_compute.cu.
• PR #3274 - Reinit counters synchronously in reinit_counters test
• PR #3273 - Splitting tests to avoid compiler OOM
• PR #3271 - Remove leftover code from context_generic_context.hpp
• PR #3269 - Fix bulk_construct with count = 0
• PR #3268 - Replace constexpr with HPX_CXX14_CONSTEXPR and HPX_CONSTEXPR
• PR #3266 - Replace boost::format with custom sprintf-based implementation
• PR #3265 - Split parallel tests on CircleCI
• PR #3262 - Making sure documentation correctly links to source files
• PR #3261 - Apex refactoring fix rebind
• PR #3260 - Isolate performance counter parser into a separate TU
• PR #3256 - Post 1.1.0 version bumps

1168 https://github.com/STEllAR-GROUP/hpx/pull/3289
1169 https://github.com/STEllAR-GROUP/hpx/pull/3288
1170 https://github.com/STEllAR-GROUP/hpx/pull/3287
1171 https://github.com/STEllAR-GROUP/hpx/pull/3286
1172 https://github.com/STEllAR-GROUP/hpx/pull/3284
1173 https://github.com/STEllAR-GROUP/hpx/pull/3283
1174 https://github.com/STEllAR-GROUP/hpx/pull/3282
1175 https://github.com/STEllAR-GROUP/hpx/pull/3281
1176 https://github.com/STEllAR-GROUP/hpx/pull/3279
1177 https://github.com/STEllAR-GROUP/hpx/pull/3278
1178 https://github.com/STEllAR-GROUP/hpx/pull/3277
1179 https://github.com/STEllAR-GROUP/hpx/pull/3276
1180 https://github.com/STEllAR-GROUP/hpx/pull/3274
1181 https://github.com/STEllAR-GROUP/hpx/pull/3273
1182 https://github.com/STEllAR-GROUP/hpx/pull/3271
1183 https://github.com/STEllAR-GROUP/hpx/pull/3269
1184 https://github.com/STEllAR-GROUP/hpx/pull/3268
1185 https://github.com/STEllAR-GROUP/hpx/pull/3266
1186 https://github.com/STEllAR-GROUP/hpx/pull/3265
1187 https://github.com/STEllAR-GROUP/hpx/pull/3262
1188 https://github.com/STEllAR-GROUP/hpx/pull/3261
1189 https://github.com/STEllAR-GROUP/hpx/pull/3260
1190 https://github.com/STEllAR-GROUP/hpx/pull/3256

Chapter 2. What’s so special about HPX?
• PR #3254[1191] - Adding trait for actions allowing to make runtime decision on whether to execute it directly
• PR #3253[1192] - Bump minimal supported Boost to 1.58.0
• PR #3251[1193] - Adds new feature: changing interval used in interval_timer (issue 3244)
• PR #3239[1194] - Changing std::rand() to a better inbuilt PRNG generator.
• PR #3234[1195] - Disable background thread when networking is off
• PR #3232[1196] - Clean up suspension tests
• PR #3230[1197] - Add optional scheduler mode parameter to create_thread_pool function
• PR #3226[1198] - Allow suspension also on static schedulers
• PR #3163[1199] - libfabric parcelport w/o HPX_PARCELPORT_LIBFABRIC_ENDPOINT_RDM
• PR #3036[1200] - Switching to CircleCI 2.0

2.11.6 HPX V1.1.0 (Mar 24, 2018)

General changes

Here are some of the main highlights and changes for this release (in no particular order):

• We have changed the way HPX manages the processing units on a node. We do not longer implicitly
  bind all available cores to a single thread pool. The user has now full control over what processing units
  are bound to what thread pool, each with a separate scheduler. It is now also possible to create your
  own scheduler implementation and control what processing units this scheduler should use. We added the
  hpx::resource::partitioner that manages all available processing units and assigns resources to the
  used thread pools. Thread pools can be now be suspended/resumed independently. This functionality helps in
  running HPX concurrently to code that is directly relying on OpenMP[1201] and/or MPI[1202].

• We have continued to implement various parallel algorithms. HPX now almost completely implements all of the
  parallel algorithms as specified by the C++17 Standard[1203]. We have also continued to implement these algo-
  rithms for the distributed use case (for segmented data structures, such as hpx::partitioned_vector).

• Added a compatibility layer for std::thread, std::mutex, and std::condition_variable allowing for the code to use those facilities where available and to fall back to the corresponding Boost facilities otherwise. The CMake[1204] configuration option -DHPX_WITH_THREAD_COMPATIBILITY=On can be used to force using the Boost equivalents.

• The parameter sequence for the hpx::parallel::transform_inclusive_scan overload taking one iterator range has changed (again) to match the changes this algorithm has undergone while being moved to C++17. The old overloads can be still enabled at configure time by passing
  -DHPX_WITH_TRANSFORM_REDUCE_COMPATIBILITY=On to CMake[1205].

[1191] https://github.com/STEllAR-GROUP/hpx/pull/3254
[1192] https://github.com/STEllAR-GROUP/hpx/pull/3253
[1193] https://github.com/STEllAR-GROUP/hpx/pull/3251
[1194] https://github.com/STEllAR-GROUP/hpx/pull/3239
[1195] https://github.com/STEllAR-GROUP/hpx/pull/3234
[1196] https://github.com/STEllAR-GROUP/hpx/pull/3232
[1197] https://github.com/STEllAR-GROUP/hpx/pull/3230
[1198] https://github.com/STEllAR-GROUP/hpx/pull/3228
[1199] https://github.com/STEllAR-GROUP/hpx/pull/3163
[1200] https://github.com/STEllAR-GROUP/hpx/pull/3036
[1201] https://openmp.org/wp/
[1203] http://www.open-std.org/jtc1/sc22/wg21
[1204] https://www.cmake.org
[1205] https://www.cmake.org
• The parameter sequence for the hpx::parallel::inclusive_scan overload taking one iterator range has changed to match the changes this algorithm has undergone while being moved to C++17. The old overloads can be still enabled at configure time by passing -DHPX_WITH_INCLUSIVE_SCAN_COMPATIBILITY=On to CMake.

• Added a helper facility hpx::local_new which is equivalent to hpx::new_ except that it creates components locally only. As a consequence, the used component constructor may accept non-serializable argument types and/or non-const references or pointers.

• Removed the (broken) component type hpx::lcos::queue<T>. The old type is still available at configure time by passing -DHPX_WITH_QUEUE_COMPATIBILITY=On to CMake.

• The parallel algorithms adopted for C++17 restrict the iterator categories usable with those to at least forward iterators. Our implementation of the parallel algorithms was supporting input iterators (and output iterators) as well by simply falling back to sequential execution. We have now made our implementations conforming by requiring at least forward iterators. In order to enable the old behavior use the the compatibility option -DHPX_WITH_ALGORITHM_INPUT_ITERATOR_SUPPORT=On on the CMake command line.

• We have added the functionalities allowing for LCOs being implemented using (simple) components. Before LCOs had to always be implemented using managed components.

• User defined components don’t have to be default-constructible anymore. Return types from actions don’t have to be default-constructible anymore either. Our serialization layer now in general supports non-default-constructible types.

• We have added a new launch policy hpx::launch::lazy that allows to defer the decision on what launch policy to use to the point of execution. This policy is initialized with a function (object) that – when invoked – is expected to produce the desired launch policy.

Breaking changes

• We have dropped support for the gcc compiler version V4.8. The minimal gcc version we now test on is gcc V4.9. The minimally required version of CMake is now V3.3.2.

• We have dropped support for the Visual Studio 2013 compiler version. The minimal Visual Studio version we now test on is Visual Studio 2015.5.

• We have dropped support for the Boost V1.51-V1.54. The minimal version of Boost we now test is Boost V1.55.

• We have dropped support for the hpx::util::unwrapped API. hpx::util::unwrapped will stay functional to some degree, until it finally gets removed in a later version of HPX. The functional usage of hpx::util::unwrapped should be changed to the new hpx::util::unwrapping function whereas the immediate usage should be replaced to hpx::util::unwrap.

• The performance counter names referring to properties as exposed by the threading subsystem have changes as those now additionally have to specify the thread-pool. See the corresponding documentation for more details.

• The overloads of hpx::async that invoke an action do not perform implicit unwrapping of the returned future anymore in case the invoked function does return a future in the first place. In this case hpx::async now returns a hpx::future<future<T>> making its behavior conforming to its local counterpart.

• We have replaced the use of boost::exception_ptr in our APIs with the equivalent std::exception_ptr. Please change your codes accordingly. No compatibility settings are provided.

• We have removed the compatibility settings for HPX_WITH_COLOCATED_BACKWARDS_COMPATIBILITY and HPX_WITH_COMPONENT_GET_GID_COMPATIBILITY as their life-cycle has reached its end.

• We have removed the experimental thread schedulers hierarchy_scheduler, periodic_priority_scheduler and throttling_scheduler in an effort to clean up and consolidate our thread schedulers.

1206 https://www.cmake.org
1207 https://www.cmake.org
Bug fixes (closed tickets)

Here is a list of the important tickets we closed for this release.

- PR #3250\(^{1208}\) - Apex refactoring with guids
- PR #3249\(^{1209}\) - Updating People.qbk
- PR #3246\(^{1210}\) - Assorted fixes for CUDA
- PR #3245\(^{1211}\) - Apex refactoring with guids
- PR #3242\(^{1212}\) - Modify task counting in thread_queue.hpp
- PR #3240\(^{1213}\) - Fixed typos
- PR #3238\(^{1214}\) - Readding accidentally removed std::abort
- PR #3237\(^{1215}\) - Adding Pipeline example
- PR #3236\(^{1216}\) - Fixing memory_block
- PR #3233\(^{1217}\) - Make schedule_thread take suspended threads into account
- Issue #3226\(^{1218}\) - memory_block is breaking, signaling SIGSEGV on a thread on creation and freeing
- PR #3225\(^{1219}\) - Applying quick fix for hwloc-2.0
- Issue #3224\(^{1220}\) - HPX counters crashing the application
- PR #3223\(^{1221}\) - Fix returns when setting config entries
- Issue #3222\(^{1222}\) - Errors linking libhpx.so
- Issue #3221\(^{1223}\) - HPX on Mac OS X with HWLoc 2.0.0 fails to run
- PR #3216\(^{1224}\) - Reorder a variadic array to satisfy VS 2017 15.6
- PR #3214\(^{1225}\) - Changed prerequisites.qbk to avoid confusion while building boost
- PR #3213\(^{1226}\) - Relax locks for thread suspension to avoid holding locks when yielding
- PR #3212\(^{1227}\) - Fix check in sequenced_executor test
- PR #3211\(^{1228}\) - Use preinit_array to set argc/argv in init_globally example
- PR #3210\(^{1229}\) - Adapted parallel::[search | search_n] for Ranges TS (see #1668)

\(^{1208}\) https://github.com/STEllAR-GROUP/hpx/pull/3250
\(^{1209}\) https://github.com/STEllAR-GROUP/hpx/pull/3249
\(^{1210}\) https://github.com/STEllAR-GROUP/hpx/pull/3246
\(^{1211}\) https://github.com/STEllAR-GROUP/hpx/pull/3245
\(^{1212}\) https://github.com/STEllAR-GROUP/hpx/pull/3242
\(^{1213}\) https://github.com/STEllAR-GROUP/hpx/pull/3240
\(^{1214}\) https://github.com/STEllAR-GROUP/hpx/pull/3238
\(^{1215}\) https://github.com/STEllAR-GROUP/hpx/pull/3237
\(^{1216}\) https://github.com/STEllAR-GROUP/hpx/pull/3236
\(^{1217}\) https://github.com/STEllAR-GROUP/hpx/pull/3233
\(^{1218}\) https://github.com/STEllAR-GROUP/hpx/issues/3226
\(^{1219}\) https://github.com/STEllAR-GROUP/hpx/pull/3225
\(^{1220}\) https://github.com/STEllAR-GROUP/hpx/issues/3224
\(^{1221}\) https://github.com/STEllAR-GROUP/hpx/pull/3223
\(^{1222}\) https://github.com/STEllAR-GROUP/hpx/issues/3222
\(^{1223}\) https://github.com/STEllAR-GROUP/hpx/issues/3221
\(^{1224}\) https://github.com/STEllAR-GROUP/hpx/pull/3216
\(^{1225}\) https://github.com/STEllAR-GROUP/hpx/pull/3214
\(^{1226}\) https://github.com/STEllAR-GROUP/hpx/pull/3213
\(^{1227}\) https://github.com/STEllAR-GROUP/hpx/pull/3212
\(^{1228}\) https://github.com/STEllAR-GROUP/hpx/pull/3211
\(^{1229}\) https://github.com/STEllAR-GROUP/hpx/pull/3210
• PR #3209\textsuperscript{1230} - Fix locking problems during shutdown
• Issue #3208\textsuperscript{1231} - init::globally throwing a run-time error
• PR #3206\textsuperscript{1232} - Addition of new arithmetic performance counter “Count”
• PR #3205\textsuperscript{1233} - Fixing return type calculation for bulk_then_execute
• PR #3204\textsuperscript{1234} - Changing std::rand() to a better inbuilt PRNG generator
• PR #3203\textsuperscript{1235} - Resolving problems during shutdown for VS2015
• PR #3202\textsuperscript{1236} - Making sure resource partitioner is not accessed if its not valid
• PR #3201\textsuperscript{1237} - Fixing optional::swap
• Issue #3200\textsuperscript{1238} - hpx::util::optional fails
• PR #3199\textsuperscript{1239} - Fix sliding_semaphore test
• PR #3198\textsuperscript{1240} - Set pre_main status before launching run_helper
• PR #3197\textsuperscript{1241} - Update README.rst
• PR #3194\textsuperscript{1242} - parallel::[fill|fill\_n] updated for Ranges TS
• PR #3193\textsuperscript{1243} - Updating Runtime.cpp by adding correct description of Performance counters during register
• PR #3191\textsuperscript{1244} - Fix sliding_semaphore_2338 test
• PR #3190\textsuperscript{1245} - Topology improvements
• PR #3189\textsuperscript{1246} - Deleting one include of median from BOOST library to arithmetics_counter file
• PR #3188\textsuperscript{1247} - Optionnally disable printing of diagnostics during terminate
• PR #3187\textsuperscript{1248} - Suppressing cmake warning issued by cmake > V3.11
• PR #3185\textsuperscript{1249} - Remove unused scoped_unlock, unlock\_guard\_try
• PR #3184\textsuperscript{1250} - Fix nqueen example
• PR #3183\textsuperscript{1251} - Add runtime start/stop, resume/suspend and OpenMP benchmarks
• Issue #3182\textsuperscript{1252} - bulk\_then\_execute has unexpected return type/des not compile

\textsuperscript{1230} https://github.com/STEllAR-GROUP/hpx/pull/3209
\textsuperscript{1231} https://github.com/STEllAR-GROUP/hpx/issues/3208
\textsuperscript{1232} https://github.com/STEllAR-GROUP/hpx/pull/3206
\textsuperscript{1233} https://github.com/STEllAR-GROUP/hpx/pull/3205
\textsuperscript{1234} https://github.com/STEllAR-GROUP/hpx/pull/3204
\textsuperscript{1235} https://github.com/STEllAR-GROUP/hpx/pull/3203
\textsuperscript{1236} https://github.com/STEllAR-GROUP/hpx/pull/3202
\textsuperscript{1237} https://github.com/STEllAR-GROUP/hpx/pull/3201
\textsuperscript{1238} https://github.com/STEllAR-GROUP/hpx/issues/3200
\textsuperscript{1239} https://github.com/STEllAR-GROUP/hpx/pull/3199
\textsuperscript{1240} https://github.com/STEllAR-GROUP/hpx/pull/3198
\textsuperscript{1241} https://github.com/STEllAR-GROUP/hpx/pull/3197
\textsuperscript{1242} https://github.com/STEllAR-GROUP/hpx/pull/3194
\textsuperscript{1243} https://github.com/STEllAR-GROUP/hpx/pull/3193
\textsuperscript{1244} https://github.com/STEllAR-GROUP/hpx/pull/3191
\textsuperscript{1245} https://github.com/STEllAR-GROUP/hpx/pull/3190
\textsuperscript{1246} https://github.com/STEllAR-GROUP/hpx/pull/3189
\textsuperscript{1247} https://github.com/STEllAR-GROUP/hpx/pull/3188
\textsuperscript{1248} https://github.com/STEllAR-GROUP/hpx/pull/3187
\textsuperscript{1249} https://github.com/STEllAR-GROUP/hpx/pull/3185
\textsuperscript{1250} https://github.com/STEllAR-GROUP/hpx/pull/3184
\textsuperscript{1251} https://github.com/STEllAR-GROUP/hpx/pull/3183
\textsuperscript{1252} https://github.com/STEllAR-GROUP/hpx/issues/3182
• Issue #3181 - hwloc 2.0 breaks topo class and cannot be used
• Issue #3180 - Schedulers that don’t support suspend/resume are unusable
• PR #3179 - Various minor changes to support FLeCSI
• PR #3178 - Fix #3124
• PR #3177 - Removed allgather
• PR #3176 - Fixed Documentation for “using_hpx_pkgconfig”
• PR #3174 - Add hpx::iostreams::ostream overload to format_to
• PR #3172 - Fix lifo queue backend
• PR #3171 - adding the missing unset() function to cpu_mask() for case of more than 64 threads
• PR #3170 - Add cmake flag -DHPX_WITH_FAULT_TOLERANCE=ON (OFF by default)
• PR #3169 - Adapted parallel::[count,count_if] for Ranges TS (see #1668)
• PR #3168 - Changing used namespace for seq execution policy
• Issue #3167 - Update GSoC projects
• Issue #3166 - Application (Octotiger) gets stuck on hpx::finalize when only using one thread
• Issue #3165 - Compilation of parallel algorithms with HPX_WITH_DATAPAR is broken
• PR #3164 - Fixing component migration
• PR #3162 - regex_from_pattern: escape regex special characters to avoid misinterpretation
• Issue #3161 - Building HPX with hwloc 2.0.0 fails
• PR #3160 - Fixing the handling of quoted command line arguments.
• PR #3158 - Fixing a race with timed suspension (second attempt)
• PR #3157 - Revert “Fixing a race with timed suspension”
• PR #3156 - Fixing serialization of classes with incompatible serialize signature
• PR #3154 - More refactorings based on clang-tidy reports

1253 https://github.com/STEllAR-GROUP/hpx/issues/3181
1254 https://github.com/STEllAR-GROUP/hpx/issues/3180
1255 https://github.com/STEllAR-GROUP/hpx/pull/3179
1256 https://github.com/STEllAR-GROUP/hpx/pull/3178
1257 https://github.com/STEllAR-GROUP/hpx/pull/3177
1258 https://github.com/STEllAR-GROUP/hpx/pull/3176
1259 https://github.com/STEllAR-GROUP/hpx/pull/3174
1260 https://github.com/STEllAR-GROUP/hpx/pull/3172
1261 https://github.com/STEllAR-GROUP/hpx/pull/3171
1262 https://github.com/STEllAR-GROUP/hpx/pull/3170
1263 https://github.com/STEllAR-GROUP/hpx/pull/3169
1264 https://github.com/STEllAR-GROUP/hpx/pull/3168
1265 https://github.com/STEllAR-GROUP/hpx/pull/3167
1266 https://github.com/STEllAR-GROUP/hpx/pull/3166
1267 https://github.com/STEllAR-GROUP/hpx/pull/3165
1268 https://github.com/STEllAR-GROUP/hpx/pull/3164
1269 https://github.com/STEllAR-GROUP/hpx/pull/3162
1270 https://github.com/STEllAR-GROUP/hpx/pull/3161
1271 https://github.com/STEllAR-GROUP/hpx/pull/3160
1272 https://github.com/STEllAR-GROUP/hpx/pull/3158
1273 https://github.com/STEllAR-GROUP/hpx/pull/3157
1274 https://github.com/STEllAR-GROUP/hpx/pull/3156
1275 https://github.com/STEllAR-GROUP/hpx/pull/3154

2.11. Releases

917
• PR #3153\textsuperscript{1276}  - Fixing a race with timed suspension
• PR #3152\textsuperscript{1277}  - Documentation for runtime suspension
• PR #3151\textsuperscript{1278}  - Use small\_vector only from boost version 1.59 onwards
• PR #3150\textsuperscript{1279}  - Avoiding more stack overflows
• PR #3148\textsuperscript{1280}  - Refactoring component\_base and base\_action/transfer\_base\_action
• PR #3147\textsuperscript{1281}  - Move yield\_while out of detail namespace and into own file
• PR #3145\textsuperscript{1282}  - Remove a leftover of the cxx11 std array cleanup
• PR #3144\textsuperscript{1283}  - Minor changes to how actions are executed
• PR #3143\textsuperscript{1284}  - Fix stack overhead
• PR #3142\textsuperscript{1285}  - Fix typo in config.hpp
• PR #3141\textsuperscript{1286}  - Fixing small\_vector compatibility with older boost version
• PR #3140\textsuperscript{1287}  - is\_heap\_text fix
• Issue #3139\textsuperscript{1288}  - Error in is\_heap\_tests.hpp
• PR #3138\textsuperscript{1289}  - Partially reverting #3126
• PR #3137\textsuperscript{1290}  - Suspend speedup
• PR #3136\textsuperscript{1291}  - Revert “Fixing #2325”
• PR #3135\textsuperscript{1292}  - Improving destruction of threads
• Issue #3134\textsuperscript{1293}  - HPX\_SERIALIZATION\_SPLIT\_FREE does not stop compiler from looking for serialize() method
• PR #3133\textsuperscript{1294}  - Make hwloc compulsory
• PR #3132\textsuperscript{1295}  - Update CXX14 constexpr feature test
• PR #3131\textsuperscript{1296}  - Fixing #2325
• PR #3130\textsuperscript{1297}  - Avoid completion handler allocation
• PR #3129\textsuperscript{1298}  - Suspend runtime

\textsuperscript{1276} https://github.com/STEllAR-GROUP/hpx/pull/3153
\textsuperscript{1277} https://github.com/STEllAR-GROUP/hpx/pull/3152
\textsuperscript{1278} https://github.com/STEllAR-GROUP/hpx/pull/3151
\textsuperscript{1279} https://github.com/STEllAR-GROUP/hpx/pull/3150
\textsuperscript{1280} https://github.com/STEllAR-GROUP/hpx/pull/3148
\textsuperscript{1281} https://github.com/STEllAR-GROUP/hpx/pull/3147
\textsuperscript{1282} https://github.com/STEllAR-GROUP/hpx/pull/3145
\textsuperscript{1283} https://github.com/STEllAR-GROUP/hpx/pull/3144
\textsuperscript{1284} https://github.com/STEllAR-GROUP/hpx/pull/3143
\textsuperscript{1285} https://github.com/STEllAR-GROUP/hpx/pull/3142
\textsuperscript{1286} https://github.com/STEllAR-GROUP/hpx/pull/3141
\textsuperscript{1287} https://github.com/STEllAR-GROUP/hpx/pull/3140
\textsuperscript{1288} https://github.com/STEllAR-GROUP/hpx/pull/3139
\textsuperscript{1289} https://github.com/STEllAR-GROUP/hpx/pull/3138
\textsuperscript{1290} https://github.com/STEllAR-GROUP/hpx/pull/3137
\textsuperscript{1291} https://github.com/STEllAR-GROUP/hpx/pull/3136
\textsuperscript{1292} https://github.com/STEllAR-GROUP/hpx/pull/3135
\textsuperscript{1293} https://github.com/STEllAR-GROUP/hpx/issues/3134
\textsuperscript{1294} https://github.com/STEllAR-GROUP/hpx/pull/3133
\textsuperscript{1295} https://github.com/STEllAR-GROUP/hpx/pull/3132
\textsuperscript{1296} https://github.com/STEllAR-GROUP/hpx/pull/3131
\textsuperscript{1297} https://github.com/STEllAR-GROUP/hpx/pull/3130
\textsuperscript{1298} https://github.com/STEllAR-GROUP/hpx/pull/3129
- PR #3128 - Make docbook dtd and xsl path names consistent
- PR #3127 - Add hpx::start nullptr overloads
- PR #3126 - Cleaning up coroutine implementation
- PR #3125 - Replacing nullptr with hpx::threads::invalid_thread_id
- Issue #3124 - Add hello_world_component to CI builds
- PR #3123 - Add new constructor.
- PR #3122 - Fixing #3121
- Issue #3121 - HPX_SMT_PAUSE is broken on non-x86 platforms when __GNUC__ is defined
- PR #3120 - Don’t use boost::intrusive_ptr for thread_id_type
- PR #3119 - Disable default executor compatibility with V1 executors
- PR #3118 - Adding performance_counter::reinit to allow for dynamically changing counter sets
- PR #3117 - Replace uses of boost/experimental::optional with util::optional
- PR #3116 - Moving background thread APEX timer #2980
- PR #3115 - Fixing race condition in channel test
- PR #3114 - Avoid using util::function for thread function wrappers
- PR #3113 - cmake V3.10.2 has changed the variable names used for MPI
- PR #3112 - Minor fixes to exclusive_scan algorithm
- PR #3111 - Revert “fix detection of cxx11_std_atomic”
- PR #3110 - Suspend thread pool
- PR #3109 - Fixing thread scheduling when yielding a thread id
- PR #3108 - Revert “Suspend thread pool”
- PR #3107 - Remove UB from thread::id relational operators
- PR #3106 - Add cmake test for std::decay_t to fix cuda build

https://github.com/STEllAR-GROUP/hpx/pull/3128
https://github.com/STEllAR-GROUP/hpx/pull/3127
https://github.com/STEllAR-GROUP/hpx/pull/3126
https://github.com/STEllAR-GROUP/hpx/pull/3125
https://github.com/STEllAR-GROUP/hpx/issues/3124
https://github.com/STEllAR-GROUP/hpx/pull/3123
https://github.com/STEllAR-GROUP/hpx/pull/3122
https://github.com/STEllAR-GROUP/hpx/pull/3121
https://github.com/STEllAR-GROUP/hpx/pull/3120
https://github.com/STEllAR-GROUP/hpx/pull/3119
https://github.com/STEllAR-GROUP/hpx/pull/3118
https://github.com/STEllAR-GROUP/hpx/pull/3117
https://github.com/STEllAR-GROUP/hpx/pull/3116
https://github.com/STEllAR-GROUP/hpx/pull/3115
https://github.com/STEllAR-GROUP/hpx/pull/3114
https://github.com/STEllAR-GROUP/hpx/pull/3113
https://github.com/STEllAR-GROUP/hpx/pull/3112
https://github.com/STEllAR-GROUP/hpx/pull/3111
https://github.com/STEllAR-GROUP/hpx/pull/3110
https://github.com/STEllAR-GROUP/hpx/pull/3109
https://github.com/STEllAR-GROUP/hpx/pull/3108
https://github.com/STEllAR-GROUP/hpx/pull/3107
https://github.com/STEllAR-GROUP/hpx/pull/3106
• PR #3105 - Fixing refcount for async traversal frame
• PR #3104 - Local execution of direct actions is now actually performed directly
• PR #3103 - Adding support for generic counter_raw_values performance counter type
• Issue #3102 - Introduce generic performance counter type returning an array of values
• PR #3101 - Revert “Adapting stack overhead limit for gcc 4.9”
• PR #3100 - Fix #3068 (condition_variable deadlock)
• PR #3099 - Fixing lock held during suspension in papi counter component
• PR #3098 - Unbreak broadcast_wait_for_2822 test
• PR #3097 - Adapting stack overhead limit for gcc 4.9
• PR #3096 - fix detection of cxx11_std_atomic
• PR #3095 - Add ciso646 header to get _LIBCPP_VERSION for testing inplace merge
• PR #3094 - Relax atomic operations on performance counter values
• PR #3093 - Short-circuit all_of/any_of/none_of instantiations
• PR #3092 - Take advantage of C++14 lambda capture initialization syntax, where possible
• PR #3091 - Remove more references to Boost from logging code
• PR #3090 - Unify use of yield/yield_k
• PR #3089 - Fix a strange thing in parallel::detail::handle_exception. (Fix #2834.)
• Issue #3088 - A strange thing in parallel::sort.
• PR #3087 - Fixing assertion in default_distribution_policy
• PR #3086 - Implement parallel::remove and parallel::remove_if
• PR #3085 - Addressing breaking changes in Boost V1.66
• PR #3084 - Ignore build warnings round 2
• PR #3083 - Fix typo HPX_WITH_MM_PREFECTH

https://github.com/STEllAR-GROUP/hpx/pull/3105
https://github.com/STEllAR-GROUP/hpx/pull/3104
https://github.com/STEllAR-GROUP/hpx/pull/3103
https://github.com/STEllAR-GROUP/hpx/issues/3102
https://github.com/STEllAR-GROUP/hpx/pull/3101
https://github.com/STEllAR-GROUP/hpx/pull/3100
https://github.com/STEllAR-GROUP/hpx/pull/3099
https://github.com/STEllAR-GROUP/hpx/pull/3098
https://github.com/STEllAR-GROUP/hpx/pull/3097
https://github.com/STEllAR-GROUP/hpx/pull/3096
https://github.com/STEllAR-GROUP/hpx/pull/3095
https://github.com/STEllAR-GROUP/hpx/pull/3094
https://github.com/STEllAR-GROUP/hpx/pull/3093
https://github.com/STEllAR-GROUP/hpx/pull/3092
https://github.com/STEllAR-GROUP/hpx/pull/3091
https://github.com/STEllAR-GROUP/hpx/pull/3090
https://github.com/STEllAR-GROUP/hpx/pull/3089
https://github.com/STEllAR-GROUP/hpx/issues/3088
https://github.com/STEllAR-GROUP/hpx/pull/3087
https://github.com/STEllAR-GROUP/hpx/pull/3086
https://github.com/STEllAR-GROUP/hpx/pull/3085
https://github.com/STEllAR-GROUP/hpx/pull/3084
https://github.com/STEllAR-GROUP/hpx/pull/3083
• PR #3081 - Pre-decay template arguments early
• PR #3080 - Suspend thread pool
• PR #3079 - Ignore build warnings
• PR #3078 - Don’t test inplace_merge with libc++
• PR #3076 - Fixing 3075: Part 1
• PR #3074 - Fix more build warnings
• PR #3073 - Suspend thread cleanup
• PR #3072 - Change existing symbol_namespace::iterate to return all data instead of invoking a callback
• PR #3071 - Fixing pack_traversal_async test
• PR #3070 - Fix dynamic_counters_loaded_1508 test by adding dependency to memory_component
• PR #3069 - Fix scheduling loop exit
• Issue #3068 - hpx::lcos::condition_variable could be suspect to deadlocks
• PR #3067 - #ifdef out random_shuffle deprecated in later c++
• PR #3066 - Make coalescing test depend on coalescing library to ensure it gets built
• PR #3065 - Workaround for minimal_timed_async_executor_test compilation failures, attempts to copy a deferred call (in unevaluated context)
• PR #3064 - Fixing wrong condition in wrapper_heap
• PR #3062 - Fix exception handling for execution::seq
• PR #3061 - Adapt MSVC C++ mode handling to VS15.5
• PR #3060 - Fix compiler problem in MSVC release mode
• PR #3059 - Fixing #2931
• Issue #3058 - minimal_timed_async_executor_test_exe fails to compile on master (d6f505c)
• PR #3057 - Fix stable_merge_2964 compilation problems
• PR #3056 - Fix some build warnings caused by unused variables/unnecessary tests

1345 https://github.com/STEllAR-GROUP/hpx/pull/3081
1346 https://github.com/STEllAR-GROUP/hpx/pull/3080
1347 https://github.com/STEllAR-GROUP/hpx/pull/3079
1348 https://github.com/STEllAR-GROUP/hpx/pull/3078
1349 https://github.com/STEllAR-GROUP/hpx/pull/3076
1350 https://github.com/STEllAR-GROUP/hpx/pull/3074
1351 https://github.com/STEllAR-GROUP/hpx/pull/3073
1352 https://github.com/STEllAR-GROUP/hpx/pull/3072
1353 https://github.com/STEllAR-GROUP/hpx/pull/3071
1354 https://github.com/STEllAR-GROUP/hpx/pull/3070
1355 https://github.com/STEllAR-GROUP/hpx/pull/3069
1356 https://github.com/STEllAR-GROUP/hpx/pull/3068
1357 https://github.com/STEllAR-GROUP/hpx/pull/3067
1358 https://github.com/STEllAR-GROUP/hpx/pull/3066
1359 https://github.com/STEllAR-GROUP/hpx/pull/3065
1360 https://github.com/STEllAR-GROUP/hpx/pull/3064
1361 https://github.com/STEllAR-GROUP/hpx/pull/3062
1362 https://github.com/STEllAR-GROUP/hpx/pull/3061
1363 https://github.com/STEllAR-GROUP/hpx/pull/3060
1364 https://github.com/STEllAR-GROUP/hpx/pull/3059
1365 https://github.com/STEllAR-GROUP/hpx/pull/3058
1366 https://github.com/STEllAR-GROUP/hpx/pull/3057
1367 https://github.com/STEllAR-GROUP/hpx/pull/3056

2.11. Releases
• PR #3055 - Update documentation for running tests
• Issue #3054 - Assertion failure when using bulk hpx::new_ in asynchronous mode
• PR #3052 - Do not bind test running to cmake test build rule
• PR #3051 - Fix HPX-Qt interaction in Qt example.
• Issue #3048 - nqueen example fails occasionally
• PR #3047 - Fixing #3044
• PR #3046 - Add OS thread suspension
• PR #3042 - PyCicle - first attempt at a build tool for checking PR’s
• PR #3041 - Fix a problem about asynchronous execution of parallel::merge and parallel::partition.
• PR #3040 - Fix a mistake about exception handling in asynchronous execution of scan_partitioner.
• PR #3039 - Consistently use executors to schedule work
• PR #3038 - Fixing local direct function execution and lambda actions perfect forwarding
• PR #3035 - Make parallel unit test names match build target/folder names
• PR #3033 - Fix setting of default build type
• Issue #3032 - Fix partitioner arg copy found in #2982
• Issue #3031 - Errors linking libhp.so due to missing references (master branch, commit 6679a8882)
• PR #3030 - Revert “implement executor then interface with && forwarding reference”
• PR #3029 - Run CI inspect checks before building
• PR #3028 - Added range version of parallel::move
• Issue #3027 - Implement all scheduling APIs in terms of executors
• PR #3026 - Implement executor then interface with && forwarding reference
• PR #3025 - Fix typo unitialized to uninitialized
• PR #3024 - Inspect fixes

1368 https://github.com/STEllAR-GROUP/hpx/pull/3055
1369 https://github.com/STEllAR-GROUP/hpx/issues/3054
1370 https://github.com/STEllAR-GROUP/hpx/pull/3052
1371 https://github.com/STEllAR-GROUP/hpx/pull/3051
1372 https://github.com/STEllAR-GROUP/hpx/issues/3048
1373 https://github.com/STEllAR-GROUP/hpx/pull/3047
1374 https://github.com/STEllAR-GROUP/hpx/pull/3046
1375 https://github.com/STEllAR-GROUP/hpx/pull/3042
1376 https://github.com/STEllAR-GROUP/hpx/pull/3041
1377 https://github.com/STEllAR-GROUP/hpx/pull/3040
1378 https://github.com/STEllAR-GROUP/hpx/pull/3039
1379 https://github.com/STEllAR-GROUP/hpx/pull/3038
1380 https://github.com/STEllAR-GROUP/hpx/pull/3035
1381 https://github.com/STEllAR-GROUP/hpx/pull/3033
1382 https://github.com/STEllAR-GROUP/hpx/issues/3032
1383 https://github.com/STEllAR-GROUP/hpx/issues/3031
1384 https://github.com/STEllAR-GROUP/hpx/pull/3030
1385 https://github.com/STEllAR-GROUP/hpx/pull/3029
1386 https://github.com/STEllAR-GROUP/hpx/pull/3028
1387 https://github.com/STEllAR-GROUP/hpx/issues/3027
1388 https://github.com/STEllAR-GROUP/hpx/pull/3026
1389 https://github.com/STEllAR-GROUP/hpx/pull/3025
1390 https://github.com/STEllAR-GROUP/hpx/pull/3024
- PR #3023\textsuperscript{1391} - P0356 Simplified partial function application
- PR #3022\textsuperscript{1392} - Master fixes
- PR #3021\textsuperscript{1393} - Segfault fix
- PR #3020\textsuperscript{1394} - Disable command-line aliasing for applications that use user_main
- PR #3019\textsuperscript{1395} - Adding enable_elasticity option to pool configuration
- PR #3018\textsuperscript{1396} - Fix stack overflow detection configuration in header files
- PR #3017\textsuperscript{1397} - Speed up local action execution
- PR #3016\textsuperscript{1398} - Unify stack-overflow detection options, remove reference to libsigsegv
- PR #3015\textsuperscript{1399} - Speeding up accessing the resource partitioner and the topology info
- Issue #3014\textsuperscript{1400} - HPX does not compile on POWER8 with gcc 5.4
- Issue #3013\textsuperscript{1401} - hello_world occasionally prints multiple lines from a single OS-thread
- PR #3012\textsuperscript{1402} - Silence warning about casting away qualifiers in itt_notify.hpp
- PR #3011\textsuperscript{1403} - Fix cpuset leak in hwloc_topology_info.cpp
- PR #3010\textsuperscript{1404} - Remove useless decay_copy
- PR #3009\textsuperscript{1405} - Fixing 2996
- PR #3008\textsuperscript{1406} - Remove unused internal function
- PR #3007\textsuperscript{1407} - Fixing wrapper_heap alignment problems
- Issue #3006\textsuperscript{1408} - hwloc memory leak
- PR #3004\textsuperscript{1409} - Silence C4251 (needs to have dll-interface) for future_data_void
- Issue #3003\textsuperscript{1410} - Suspension of runtime
- PR #3001\textsuperscript{1411} - Attempting to avoid data races in async_traversal while evaluating dataflow()
- PR #3000\textsuperscript{1412} - Adding hpx::util::optional as a first step to replace experimental::optional
- PR #2998\textsuperscript{1413} - Cleanup up and Fixing component creation and deletion
- Issue #2996 - Build fails with HPX_WITH_HWLOC=OFF
- PR #2995 - Push more future\_data functionality to source file
- PR #2994 - WIP: Fix throttle test
- PR #2993 - Making sure --hpX:help does not throw for required (but missing) arguments
- PR #2992 - Adding non-blocking (on destruction) service executors
- Issue #2991 - run\_as\_os\_thread locks up
- Issue #2990 - --help will not work until all required options are provided
- PR #2989 - Improve error messages caused by misuse of dataflow
- PR #2988 - Improve error messages caused by misuse of .then
- Issue #2987 - stack overflow detection producing false positives
- PR #2986 - Deduplicate non-dependent thread\_info logging types
- PR #2985 - Adapted parallel::\{all\_of|any\_of|none\_of\} for Ranges TS (see #1668)
- PR #2984 - Refactor one\_size\_heap code to simplify code
- PR #2983 - Fixing local\_new\_component
- PR #2982 - Clang tidy
- PR #2981 - Simplify allocator rebinding in pack traversal
- PR #2979 - Fixing integer overflows
- PR #2978 - Implement parallel::inplace\_merge
- Issue #2977 - Make hwloc compulsory instead of optional
- PR #2976 - Making sure client\_base instance that registered the component does not unregister it when being destructed
- PR #2975 - Change version of pulled APEX to master
- PR #2974 - Fix domain not being freed at the end of scheduling loop
- PR #2973 - Fix small typos

[1414] https://github.com/STEllAR-GROUP/hpx/issues/2996
[1415] https://github.com/STEllAR-GROUP/hpx/pull/2995
[1416] https://github.com/STEllAR-GROUP/hpx/pull/2994
[1417] https://github.com/STEllAR-GROUP/hpx/pull/2993
[1418] https://github.com/STEllAR-GROUP/hpx/pull/2992
[1419] https://github.com/STEllAR-GROUP/hpx/issues/2991
[1420] https://github.com/STEllAR-GROUP/hpx/issues/2990
[1421] https://github.com/STEllAR-GROUP/hpx/pull/2989
[1422] https://github.com/STEllAR-GROUP/hpx/pull/2988
[1423] https://github.com/STEllAR-GROUP/hpx/issues/2987
[1424] https://github.com/STEllAR-GROUP/hpx/pull/2986
[1425] https://github.com/STEllAR-GROUP/hpx/pull/2985
[1426] https://github.com/STEllAR-GROUP/hpx/pull/2984
[1427] https://github.com/STEllAR-GROUP/hpx/pull/2983
[1428] https://github.com/STEllAR-GROUP/hpx/pull/2982
[1429] https://github.com/STEllAR-GROUP/hpx/pull/2981
[1430] https://github.com/STEllAR-GROUP/hpx/pull/2979
[1431] https://github.com/STEllAR-GROUP/hpx/pull/2978
[1432] https://github.com/STEllAR-GROUP/hpx/issues/2977
[1433] https://github.com/STEllAR-GROUP/hpx/pull/2976
[1434] https://github.com/STEllAR-GROUP/hpx/pull/2975
[1435] https://github.com/STEllAR-GROUP/hpx/pull/2974
[1436] https://github.com/STEllAR-GROUP/hpx/pull/2973
• PR #2972 - Adding uintstd.h header
• PR #2971 - Fall back to creating local components using local_new
• PR #2970 - Improve is_tuple_like trait
• PR #2969 - Fix HPX_WITH_MORE_THAN_64_THREADS default value
• PR #2968 - Cleaning up dataflow overload set
• PR #2967 - Make parallel::merge is stable. (Fix #2964.)
• PR #2966 - Fixing a couple of held locks during exception handling
• PR #2965 - Adding missing #include
• Issue #2964 - parallel merge is not stable
• PR #2963 - Making sure any function object passed to dataflow is released after being invoked
• PR #2962 - Partially reverting #2891
• PR #2961 - Attempt to fix the gcc 4.9 problem with the async pack traversal
• Issue #2959 - Program terminates during error handling
• Issue #2958 - HPXPLAIN_ACTION breaks due to missing include
• PR #2957 - Fixing errors generated by mixing different attribute syntaxes
• Issue #2956 - Mixing attribute syntaxes leads to compiler errors
• Issue #2955 - Fix OS-Thread throttling
• PR #2953 - Making sure any hpx.os_threads=N supplied through a --hpx::config file is taken into account
• PR #2952 - Removing wrong call to cleanup_terminated_locked
• PR #2951 - Revert “Make sure the function vtables are initialized before use”
• PR #2950 - Fix a namespace compilation error when some schedulers are disabled
• Issue #2949 - master branch giving lockups on shutdown
• Issue #2947 - hpx.ini is not used correctly at initialization

1437 https://github.com/STEllAR-GROUP/hpx/pull/2972
1438 https://github.com/STEllAR-GROUP/hpx/pull/2971
1439 https://github.com/STEllAR-GROUP/hpx/pull/2970
1440 https://github.com/STEllAR-GROUP/hpx/pull/2969
1441 https://github.com/STEllAR-GROUP/hpx/pull/2968
1442 https://github.com/STEllAR-GROUP/hpx/pull/2967
1443 https://github.com/STEllAR-GROUP/hpx/pull/2966
1444 https://github.com/STEllAR-GROUP/hpx/pull/2965
1445 https://github.com/STEllAR-GROUP/hpx/issues/2964
1446 https://github.com/STEllAR-GROUP/hpx/pull/2963
1447 https://github.com/STEllAR-GROUP/hpx/pull/2962
1448 https://github.com/STEllAR-GROUP/hpx/pull/2961
1449 https://github.com/STEllAR-GROUP/hpx/issues/2959
1450 https://github.com/STEllAR-GROUP/hpx/issues/2958
1451 https://github.com/STEllAR-GROUP/hpx/pull/2957
1452 https://github.com/STEllAR-GROUP/hpx/issues/2956
1453 https://github.com/STEllAR-GROUP/hpx/issues/2955
1454 https://github.com/STEllAR-GROUP/hpx/pull/2953
1455 https://github.com/STEllAR-GROUP/hpx/pull/2952
1456 https://github.com/STEllAR-GROUP/hpx/pull/2951
1457 https://github.com/STEllAR-GROUP/hpx/pull/2950
1458 https://github.com/STEllAR-GROUP/hpx/issues/2949
1459 https://github.com/STEllAR-GROUP/hpx/issues/2947
• PR #2946 - Adding explicit feature test for thread_local
• PR #2945 - Make sure the function vtables are initialized before use
• PR #2944 - Attempting to solve affinity problems on CircleCI
• PR #2943 - Changing channel actions to be direct
• PR #2942 - Adding split_future for std::vector
• PR #2941 - Add a feature test to test for CXX11 override
• Issue #2940 - Add split_future for future<vector<T>>
• PR #2939 - Making error reporting during problems with setting affinity masks more verbose
• PR #2938 - Fix this various executors
• PR #2937 - Fix some typos in documentation
• PR #2934 - Remove the need for “complete” SFINAE checks
• PR #2933 - Making sure parallel::for_loop is executed in parallel if requested
• PR #2932 - Classify chunk_size_iterator to input iterator tag. (Fix #2866)
• Issue #2931 - hpx:help triggers unusual error with clang build
• PR #2930 - Add #include files needed to set _POSIX_VERSION for debug check
• PR #2929 - Fix a couple of deprecated c++ features
• PR #2928 - Fixing execution parameters
• Issue #2927 - CMake warning: … cycle in constraint graph
• PR #2926 - Default pool rename
• Issue #2925 - Default pool cannot be renamed
• Issue #2924 - hpx:attach-debugger=startup does not work any more
• PR #2923 - Alloc membind
• PR #2922 - This fixes CircleCI errors when running with –hpx:bind=none

https://github.com/STEllAR-GROUP/hpx/pull/2946
https://github.com/STEllAR-GROUP/hpx/pull/2945
https://github.com/STEllAR-GROUP/hpx/pull/2944
https://github.com/STEllAR-GROUP/hpx/pull/2943
https://github.com/STEllAR-GROUP/hpx/pull/2942
https://github.com/STEllAR-GROUP/hpx/pull/2941
https://github.com/STEllAR-GROUP/hpx/issues/2940
https://github.com/STEllAR-GROUP/hpx/pull/2939
https://github.com/STEllAR-GROUP/hpx/pull/2938
https://github.com/STEllAR-GROUP/hpx/pull/2937
https://github.com/STEllAR-GROUP/hpx/pull/2934
https://github.com/STEllAR-GROUP/hpx/pull/2933
https://github.com/STEllAR-GROUP/hpx/pull/2932
https://github.com/STEllAR-GROUP/hpx/issues/2931
https://github.com/STEllAR-GROUP/hpx/pull/2930
https://github.com/STEllAR-GROUP/hpx/pull/2929
https://github.com/STEllAR-GROUP/hpx/pull/2928
https://github.com/STEllAR-GROUP/hpx/issues/2927
https://github.com/STEllAR-GROUP/hpx/pull/2926
https://github.com/STEllAR-GROUP/hpx/issues/2925
https://github.com/STEllAR-GROUP/hpx/issues/2924
https://github.com/STEllAR-GROUP/hpx/pull/2923
https://github.com/STEllAR-GROUP/hpx/pull/2922
- PR #2921\textsuperscript{1483} - Custom pool executor was missing priority and stacksize options
- PR #2920\textsuperscript{1484} - Adding test to trigger problem reported in #2916
- PR #2919\textsuperscript{1485} - Make sure the resource\_partitioner is properly destructed on hpx::finalize
- Issue #2918\textsuperscript{1486} - hpx::init calls wrong (first) callback when called multiple times
- PR #2917\textsuperscript{1487} - Adding util::checkpoint
- Issue #2916\textsuperscript{1488} - Weird runtime failures when using a channel and chained continuations
- PR #2915\textsuperscript{1489} - Introduce executor parameters customization points
- Issue #2914\textsuperscript{1490} - Task assignment to current Pool has unintended consequences
- PR #2913\textsuperscript{1491} - Fix rp hang
- PR #2912\textsuperscript{1492} - Update contributors
- PR #2911\textsuperscript{1493} - Fixing CUDA problems
- PR #2910\textsuperscript{1494} - Improve error reporting for process component on POSIX systems
- PR #2909\textsuperscript{1495} - Fix typo in include path
- PR #2908\textsuperscript{1496} - Use proper container according to iterator tag in benchmarks of parallel algorithms
- PR #2907\textsuperscript{1497} - Optionaly force-delete remaining channel items on close
- PR #2906\textsuperscript{1498} - Making sure generated performance counter names are correct
- Issue #2905\textsuperscript{1499} - collecting idle-rate performance counters on multiple localities produces an error
- Issue #2904\textsuperscript{1500} - build broken for Intel 17 compilers
- PR #2903\textsuperscript{1501} - Documentation Updates– Adding New People
- PR #2902\textsuperscript{1502} - Fixing service\_executor
- PR #2901\textsuperscript{1503} - Fixing partitioned\_vector creation
- PR #2900\textsuperscript{1504} - Add numa-balanced mode to hpx::bind, spread cores over numa domains
- Issue #2899\textsuperscript{1505} - hpx::bind does not have a mode that balances cores over numa domains

\textsuperscript{1483} https://github.com/STEllAR-GROUP/hpx/pull/2921
\textsuperscript{1484} https://github.com/STEllAR-GROUP/hpx/pull/2920
\textsuperscript{1485} https://github.com/STEllAR-GROUP/hpx/pull/2919
\textsuperscript{1486} https://github.com/STEllAR-GROUP/hpx/issues/2918
\textsuperscript{1487} https://github.com/STEllAR-GROUP/hpx/pull/2917
\textsuperscript{1488} https://github.com/STEllAR-GROUP/hpx/pull/2916
\textsuperscript{1489} https://github.com/STEllAR-GROUP/hpx/pull/2915
\textsuperscript{1490} https://github.com/STEllAR-GROUP/hpx/issues/2914
\textsuperscript{1491} https://github.com/STEllAR-GROUP/hpx/pull/2913
\textsuperscript{1492} https://github.com/STEllAR-GROUP/hpx/pull/2912
\textsuperscript{1493} https://github.com/STEllAR-GROUP/hpx/pull/2911
\textsuperscript{1494} https://github.com/STEllAR-GROUP/hpx/pull/2910
\textsuperscript{1495} https://github.com/STEllAR-GROUP/hpx/pull/2909
\textsuperscript{1496} https://github.com/STEllAR-GROUP/hpx/pull/2908
\textsuperscript{1497} https://github.com/STEllAR-GROUP/hpx/pull/2907
\textsuperscript{1498} https://github.com/STEllAR-GROUP/hpx/pull/2906
\textsuperscript{1499} https://github.com/STEllAR-GROUP/hpx/issues/2905
\textsuperscript{1500} https://github.com/STEllAR-GROUP/hpx/issues/2904
\textsuperscript{1501} https://github.com/STEllAR-GROUP/hpx/pull/2903
\textsuperscript{1502} https://github.com/STEllAR-GROUP/hpx/pull/2902
\textsuperscript{1503} https://github.com/STEllAR-GROUP/hpx/pull/2901
\textsuperscript{1504} https://github.com/STEllAR-GROUP/hpx/pull/2900
\textsuperscript{1505} https://github.com/STEllAR-GROUP/hpx/issues/2899
• PR #2898\textsuperscript{1506} - Adding missing #include and missing guard for optional code section
• PR #2897\textsuperscript{1507} - Removing dependency on Boost.ICL
• Issue #2896\textsuperscript{1508} - Debug build fails without -fpermissive with GCC 7.1 and Boost 1.65
• PR #2895\textsuperscript{1509} - Fixing SLURM environment parsing
• PR #2894\textsuperscript{1510} - Fix incorrect handling of compile definition with value 0
• Issue #2893\textsuperscript{1511} - Disabling schedulers causes build errors
• PR #2892\textsuperscript{1512} - added list serializer
• PR #2891\textsuperscript{1513} - Resource Partitioner Fixes
• Issue #2890\textsuperscript{1514} - Destroying a non-empty channel causes an assertion failure
• PR #2889\textsuperscript{1515} - Add check for libatomic
• PR #2888\textsuperscript{1516} - Fix compilation problems if HPX\_WITH\_ITT\_NOTIFY=ON
• PR #2887\textsuperscript{1517} - Adapt broadcast() to non-unwrapping async\langle Action\rangle
• PR #2886\textsuperscript{1518} - Replace Boost.Random with C++11 \texttt{<random>}
• Issue #2885\textsuperscript{1519} - regression in broadcast?
• Issue #2884\textsuperscript{1520} - linking \texttt{-latomic} is not portable
• PR #2883\textsuperscript{1521} - Explicitly set \texttt{-pthread} flag if available
• PR #2882\textsuperscript{1522} - Wrap boost::format uses
• Issue #2881\textsuperscript{1523} - hpx not compiling with HPX\_WITH\_ITT\_NOTIFY=On
• Issue #2880\textsuperscript{1524} - hpx::bind scatter/balanced give wrong pu masks
• PR #2878\textsuperscript{1525} - Fix incorrect pool usage masks setup in RP/thread manager
• PR #2877\textsuperscript{1526} - Require std::array by default
• PR #2875\textsuperscript{1527} - Deprecate use of BOOST\_ASSERT
• PR #2874\textsuperscript{1528} - Changed serialization of boost\_variant to use variadic templates

\textsuperscript{1506}https://github.com/STEllAR-GROUP/hpx/pull/2898
\textsuperscript{1507}https://github.com/STEllAR-GROUP/hpx/pull/2897
\textsuperscript{1508}https://github.com/STEllAR-GROUP/hpx/issues/2896
\textsuperscript{1509}https://github.com/STEllAR-GROUP/hpx/pull/2895
\textsuperscript{1510}https://github.com/STEllAR-GROUP/hpx/pull/2894
\textsuperscript{1511}https://github.com/STEllAR-GROUP/hpx/issues/2893
\textsuperscript{1512}https://github.com/STEllAR-GROUP/hpx/pull/2892
\textsuperscript{1513}https://github.com/STEllAR-GROUP/hpx/pull/2891
\textsuperscript{1514}https://github.com/STEllAR-GROUP/hpx/issues/2890
\textsuperscript{1515}https://github.com/STEllAR-GROUP/hpx/pull/2889
\textsuperscript{1516}https://github.com/STEllAR-GROUP/hpx/pull/2888
\textsuperscript{1517}https://github.com/STEllAR-GROUP/hpx/pull/2887
\textsuperscript{1518}https://github.com/STEllAR-GROUP/hpx/pull/2886
\textsuperscript{1519}https://github.com/STEllAR-GROUP/hpx/issues/2885
\textsuperscript{1520}https://github.com/STEllAR-GROUP/hpx/issues/2884
\textsuperscript{1521}https://github.com/STEllAR-GROUP/hpx/pull/2883
\textsuperscript{1522}https://github.com/STEllAR-GROUP/hpx/pull/2882
\textsuperscript{1523}https://github.com/STEllAR-GROUP/hpx/issues/2881
\textsuperscript{1524}https://github.com/STEllAR-GROUP/hpx/issues/2880
\textsuperscript{1525}https://github.com/STEllAR-GROUP/hpx/pull/2878
\textsuperscript{1526}https://github.com/STEllAR-GROUP/hpx/pull/2877
\textsuperscript{1527}https://github.com/STEllAR-GROUP/hpx/pull/2875
\textsuperscript{1528}https://github.com/STEllAR-GROUP/hpx/pull/2874
• Issue #2873 - building with parcelport_mpi fails on cori
• PR #2871 - Adding missing support for throttling scheduler
• PR #2870 - Disambiguate use of base_lco_with_value macros with channel
• Issue #2869 - Difficulty compiling HPX_REGISTER_CHANNEL_DECLARATION(double)
• PR #2868 - Removing unneeded assert
• PR #2867 - Implement parallel::unique
• Issue #2866 - The chunk_size_iterator violates multipass guarantee
• PR #2865 - Only use sched_getcpu on linux machines
• PR #2864 - Create redistribution archive for successful builds
• PR #2863 - Replace casts/assignments with hard-coded memcpy operations
• Issue #2862 - sched_getcpu not available on MacOS
• PR #2861 - Fixing unmatched header defines and recursive inclusion of threadmanager
• Issue #2860 - Master program fails with assertion ‘type == data_type_address’ failed: HPX(assertion_failure)
• Issue #2852 - Support for ARM64
• PR #2858 - Fix misplaced #if #endif’s that cause build failure without THREAD_CUMULATIVE_COUNTS
• PR #2857 - Fix some listing in documentation
• PR #2856 - Fixing component handling for lcos
• PR #2855 - Add documentation for coarrays
• PR #2854 - Support ARM64 in timestamps
• PR #2853 - Update Table 17. Non-modifying Parallel Algorithms in Documentation
• PR #2851 - Allowing for non-default-constructible component types
• PR #2850 - Enable returning future<R> from actions where R is not default-constructible

1529 https://github.com/STEllAR-GROUP/hpx/issues/2873
1530 https://github.com/STEllAR-GROUP/hpx/pull/2871
1531 https://github.com/STEllAR-GROUP/hpx/pull/2870
1532 https://github.com/STEllAR-GROUP/hpx/issues/2869
1533 https://github.com/STEllAR-GROUP/hpx/pull/2868
1534 https://github.com/STEllAR-GROUP/hpx/pull/2867
1535 https://github.com/STEllAR-GROUP/hpx/issues/2866
1536 https://github.com/STEllAR-GROUP/hpx/pull/2865
1537 https://github.com/STEllAR-GROUP/hpx/pull/2864
1538 https://github.com/STEllAR-GROUP/hpx/pull/2863
1539 https://github.com/STEllAR-GROUP/hpx/issues/2862
1540 https://github.com/STEllAR-GROUP/hpx/pull/2861
1541 https://github.com/STEllAR-GROUP/hpx/issues/2860
1542 https://github.com/STEllAR-GROUP/hpx/issues/2852
1543 https://github.com/STEllAR-GROUP/hpx/pull/2858
1544 https://github.com/STEllAR-GROUP/hpx/pull/2857
1545 https://github.com/STEllAR-GROUP/hpx/pull/2856
1546 https://github.com/STEllAR-GROUP/hpx/pull/2855
1547 https://github.com/STEllAR-GROUP/hpx/pull/2854
1548 https://github.com/STEllAR-GROUP/hpx/pull/2853
1549 https://github.com/STEllAR-GROUP/hpx/pull/2851
1550 https://github.com/STEllAR-GROUP/hpx/pull/2850
• PR #2849 - Unify serialization of non-default-constructable types
• Issue #2848 - Components have to be default constructible
• Issue #2847 - Returning a future<R> where R is not default-constructable broken
• Issue #2846 - Unify serialization of non-default-constructible types
• PR #2845 - Add Visual Studio 2015 to the tested toolchains in Appveyor
• Issue #2844 - Change the appveyor build to use the minimal required MSVC version
• Issue #2843 - multi node hello_world hangs
• PR #2842 - Correcting Spelling mistake in docs
• PR #2841 - Fix usage of std::aligned_storage
• PR #2840 - Remove constexpr from a void function
• Issue #2839 - memcp y buffer overflow: load_construct_data() and std::complex members
• Issue #2835 - constexpr functions with void return type break compilation with CUDA 8.0
• Issue #2834 - One suspicion in parallel::detail::handle_exception
• PR #2833 - Implement parallel::merge
• PR #2832 - Fix a strange thing in parallel::util::detail::handle_local_exceptions. (Fix #2818)
• PR #2830 - Break the debugger when a test failed
• Issue #2831 - parallel/executor s/execution_fwd.hpp causes compilation failure in C++11 mode.
• PR #2829 - Implement an API for asynchronous pack traversal
• PR #2828 - Split unit test builds on CircleCI to avoid timeouts
• Issue #2827 - failure to compile hello_world example with -Werror
• PR #2824 - Making sure promises are marked as started when used as continuations
• PR #2823 - Add documentation for partitioned_vector_view
• Issue #2822 - Yet another issue with wait_for similar to #2796

https://github.com/STEllAR-GROUP/hpx/pull/2849
https://github.com/STEllAR-GROUP/hpx/issues/2848
https://github.com/STEllAR-GROUP/hpx/issues/2847
https://github.com/STEllAR-GROUP/hpx/issues/2846
https://github.com/STEllAR-GROUP/hpx/pull/2845
https://github.com/STEllAR-GROUP/hpx/issues/2844
https://github.com/STEllAR-GROUP/hpx/issues/2843
https://github.com/STEllAR-GROUP/hpx/pull/2842
https://github.com/STEllAR-GROUP/hpx/pull/2841
https://github.com/STEllAR-GROUP/hpx/pull/2840
https://github.com/STEllAR-GROUP/hpx/issues/2839
https://github.com/STEllAR-GROUP/hpx/issues/2835
https://github.com/STEllAR-GROUP/hpx/pull/2834
https://github.com/STEllAR-GROUP/hpx/pull/2833
https://github.com/STEllAR-GROUP/hpx/pull/2832
https://github.com/STEllAR-GROUP/hpx/pull/2830
https://github.com/STEllAR-GROUP/hpx/issues/2831
https://github.com/STEllAR-GROUP/hpx/pull/2829
https://github.com/STEllAR-GROUP/hpx/pull/2828
https://github.com/STEllAR-GROUP/hpx/issues/2827
https://github.com/STEllAR-GROUP/hpx/pull/2824
https://github.com/STEllAR-GROUP/hpx/pull/2823
https://github.com/STEllAR-GROUP/hpx/issues/2822
• PR #2821\textsuperscript{1574} - Fix bugs and improve that about HPX\_HAVE\_CXX11\_AUTO\_RETURN\_VALUE of CMake
• PR #2820\textsuperscript{1575} - Support C++11 in benchmark codes of parallel::partition and parallel::partition\_copy
• PR #2819\textsuperscript{1576} - Fix compile errors in unit test of container version of parallel::partition
• Issue #2818\textsuperscript{1577} - A strange thing in parallel::util::detail::handle\_local\_exceptions
• Issue #2815\textsuperscript{1578} - HPX fails to compile with HPX\_WITH\_CUDA=ON and the new CUDA 9.0 RC
• Issue #2814\textsuperscript{1579} - Using ‘gmakeN’ after ‘cmake’ produces error in src/CMakeFiles/hpx.dir/runtime/agas/addressing\_service.cpp.o
• PR #2813\textsuperscript{1580} - Properly support [[noreturn]] attribute if available
• Issue #2812\textsuperscript{1581} - Compilation fails with gcc 7.1.1
• PR #2811\textsuperscript{1582} - Adding hpx::launch::lazy and support for async, dataflow, and future::then
• PR #2810\textsuperscript{1583} - Add option allowing to disable deprecation warning
• PR #2809\textsuperscript{1584} - Disable throttling scheduler if HWLOC is not found/used
• PR #2808\textsuperscript{1585} - Fix compile errors on some environments of parallel::partition
• Issue #2807\textsuperscript{1586} - Difficulty building with HPX\_WITH\_HWLOC=Off
• PR #2806\textsuperscript{1587} - Partitioned vector
• PR #2805\textsuperscript{1588} - Serializing collections with non-default constructible data
• PR #2802\textsuperscript{1589} - Fix FreeBSD 11
• Issue #2801\textsuperscript{1590} - Rate limiting techniques in io\_service
• Issue #2800\textsuperscript{1591} - New Launch Policy: async\_if
• PR #2799\textsuperscript{1592} - Fix a unit test failure on GCC in tuple\_cat
• PR #2798\textsuperscript{1593} - bump minimum required cmake to 3.0 in test
• PR #2797\textsuperscript{1594} - Making sure future::wait\_for et.al. work properly for action results
• Issue #2796\textsuperscript{1595} - wait\_for does always in “deferred” state for calls on remote localities
• Issue #2795\textsuperscript{1596} - Serialization of types without default constructor

\textsuperscript{1574} https://github.com/STEllAR-GROUP/hpx/pull/2821
\textsuperscript{1575} https://github.com/STEllAR-GROUP/hpx/pull/2820
\textsuperscript{1576} https://github.com/STEllAR-GROUP/hpx/pull/2819
\textsuperscript{1577} https://github.com/STEllAR-GROUP/hpx/issues/2818
\textsuperscript{1578} https://github.com/STEllAR-GROUP/hpx/issues/2815
\textsuperscript{1579} https://github.com/STEllAR-GROUP/hpx/issues/2814
\textsuperscript{1580} https://github.com/STEllAR-GROUP/hpx/pull/2813
\textsuperscript{1581} https://github.com/STEllAR-GROUP/hpx/issues/2812
\textsuperscript{1582} https://github.com/STEllAR-GROUP/hpx/pull/2811
\textsuperscript{1583} https://github.com/STEllAR-GROUP/hpx/pull/2810
\textsuperscript{1584} https://github.com/STEllAR-GROUP/hpx/pull/2809
\textsuperscript{1585} https://github.com/STEllAR-GROUP/hpx/pull/2808
\textsuperscript{1586} https://github.com/STEllAR-GROUP/hpx/issues/2807
\textsuperscript{1587} https://github.com/STEllAR-GROUP/hpx/pull/2806
\textsuperscript{1588} https://github.com/STEllAR-GROUP/hpx/pull/2805
\textsuperscript{1589} https://github.com/STEllAR-GROUP/hpx/pull/2802
\textsuperscript{1590} https://github.com/STEllAR-GROUP/hpx/issues/2801
\textsuperscript{1591} https://github.com/STEllAR-GROUP/hpx/issues/2800
\textsuperscript{1592} https://github.com/STEllAR-GROUP/hpx/pull/2799
\textsuperscript{1593} https://github.com/STEllAR-GROUP/hpx/pull/2798
\textsuperscript{1594} https://github.com/STEllAR-GROUP/hpx/pull/2797
\textsuperscript{1595} https://github.com/STEllAR-GROUP/hpx/issues/2796
\textsuperscript{1596} https://github.com/STEllAR-GROUP/hpx/issues/2795
• PR #2794 - Fixing test for partitioned_vector iteration
• PR #2792 - Implemented segmented find and its variations for partitioned vector
• PR #2791 - Circumvent scary warning about placement new
• PR #2790 - Fix OSX build
• PR #2789 - Resource partitioner
• PR #2788 - Adapt parallel::is_heap and parallel::is_heap_until to Ranges TS
• PR #2787 - Unwrap hotfixes
• PR #2786 - Update CMake Minimum Version to 3.3.2 (refs #2565)
• Issue #2785 - Issues with masks and cpuset
• PR #2784 - Error with reduce and transform reduce fixed
• PR #2783 - StackOverflow integration with libsigsegv
• PR #2782 - Replace boost::atomic with std::atomic (where possible)
• PR #2781 - Check for and optionally use [[deprecated]] attribute
• PR #2780 - Adding empty (but non-trivial) destructor to circumvent warnings
• PR #2779 - Exception info tweaks
• PR #2778 - Implement parallel::partition
• PR #2777 - Improve error handling in gather_here/gather_there
• PR #2776 - Fix a bug in compiler version check
• PR #2775 - Fix compilation when HPX_WITH_LOGGING is OFF
• PR #2774 - Removing dependency on Boost.Date_Time
• PR #2773 - Add sync_images() method to spmd_block class
• PR #2772 - Adding documentation for PAPI counters
• PR #2771 - Removing boost preprocessor dependency

https://github.com/STEllAR-GROUP/hpx/pull/2794
https://github.com/STEllAR-GROUP/hpx/pull/2792
https://github.com/STEllAR-GROUP/hpx/pull/2791
https://github.com/STEllAR-GROUP/hpx/pull/2790
https://github.com/STEllAR-GROUP/hpx/pull/2789
https://github.com/STEllAR-GROUP/hpx/pull/2788
https://github.com/STEllAR-GROUP/hpx/pull/2787
https://github.com/STEllAR-GROUP/hpx/pull/2786
https://github.com/STEllAR-GROUP/hpx/pull/2785
https://github.com/STEllAR-GROUP/hpx/pull/2784
https://github.com/STEllAR-GROUP/hpx/pull/2783
https://github.com/STEllAR-GROUP/hpx/pull/2782
https://github.com/STEllAR-GROUP/hpx/pull/2781
https://github.com/STEllAR-GROUP/hpx/pull/2780
https://github.com/STEllAR-GROUP/hpx/pull/2779
https://github.com/STEllAR-GROUP/hpx/pull/2778
https://github.com/STEllAR-GROUP/hpx/pull/2777
https://github.com/STEllAR-GROUP/hpx/pull/2776
https://github.com/STEllAR-GROUP/hpx/pull/2775
https://github.com/STEllAR-GROUP/hpx/pull/2774
https://github.com/STEllAR-GROUP/hpx/pull/2773
https://github.com/STEllAR-GROUP/hpx/pull/2772
https://github.com/STEllAR-GROUP/hpx/pull/2771
• PR #2770 - Adding test, fixing deadlock in config registry
• PR #2769 - Remove some other warnings and errors detected by clang 5.0
• Issue #2768 - Is there iterator tag for HPX?
• PR #2767 - Improvements to continuation annotation
• PR #2765 - gcc split stack support for HPX threads #620
• PR #2764 - Fix some uses of begin/end, remove unnecessary includes
• PR #2763 - Bump minimal Boost version to 1.55.0
• PR #2762 - hpx::partitioned_vector serializer
• PR #2761 - Adding configuration summary to cmake output and --hpx:info
• PR #2760 - Removing 1d_hydro example as it is broken
• PR #2759 - Remove various warnings detected by clang 5.0
• Issue #2757 - In case of a “raw thread” is needed per core for implementing parallel algorithm, what is good practice in HPX?
• PR #2756 - Allowing for LCOs to be simple components
• PR #2755 - Removing make_index_pack_unrolled
• PR #2754 - Implement parallel::unique_copy
• PR #2753 - Fixing detection of [[fallthrough]] attribute
• PR #2752 - New thread priority names
• PR #2751 - Replace boost::exception with proposed exception_info
• PR #2750 - Replace boost::iterator_range
• PR #2749 - Fixing hdf5 examples
• Issue #2748 - HPX fails to build with enabled hdf5 examples
• Issue #2747 - Inherited task priorities break certain DAG optimizations
• Issue #2746 - HPX segfaulting with valgrind

https://github.com/STEllAR-GROUP/hpx/pull/2770
https://github.com/STEllAR-GROUP/hpx/pull/2769
https://github.com/STEllAR-GROUP/hpx/issues/2768
https://github.com/STEllAR-GROUP/hpx/pull/2767
https://github.com/STEllAR-GROUP/hpx/pull/2765
https://github.com/STEllAR-GROUP/hpx/pull/2764
https://github.com/STEllAR-GROUP/hpx/pull/2763
https://github.com/STEllAR-GROUP/hpx/pull/2762
https://github.com/STEllAR-GROUP/hpx/pull/2761
https://github.com/STEllAR-GROUP/hpx/pull/2760
https://github.com/STEllAR-GROUP/hpx/pull/2758
https://github.com/STEllAR-GROUP/hpx/issues/2757
https://github.com/STEllAR-GROUP/hpx/pull/2756
https://github.com/STEllAR-GROUP/hpx/pull/2755
https://github.com/STEllAR-GROUP/hpx/pull/2754
https://github.com/STEllAR-GROUP/hpx/pull/2753
https://github.com/STEllAR-GROUP/hpx/pull/2752
https://github.com/STEllAR-GROUP/hpx/pull/2751
https://github.com/STEllAR-GROUP/hpx/pull/2750
https://github.com/STEllAR-GROUP/hpx/pull/2749
https://github.com/STEllAR-GROUP/hpx/issues/2748
https://github.com/STEllAR-GROUP/hpx/issues/2747
https://github.com/STEllAR-GROUP/hpx/issues/2746

2.11. Releases
• PR #2745 - Adding extended arithmetic performance counters
• PR #2744 - Adding ability to statistics counters to reset base counter
• Issue #2743 - Statistics counter does not support resetting
• PR #2742 - Making sure Vc V2 builds without additional HPX configuration flags
• PR #2741 - Deprecate unwrapped and implement unwrap and unwrapping
• PR #2740 - Coroutine stackoverflow detection for linux/posix; Issue #2408
• PR #2739 - Add files via upload
• PR #2738 - Appveyor support
• PR #2737 - Fixing 2735
• Issue #2736 - ld_hydro example doesn’t work
• Issue #2735 - partitioned_vectorSubview test failing
• PR #2734 - Add C++11 range utilities
• PR #2733 - Adapting iterator requirements for parallel algorithms
• PR #2732 - Integrate C++ Co-arrays
• PR #2731 - Adding on_migrated event handler to migratable component instances
• Issue #2729 - Add on_migrated() event handler to migratable components
• Issue #2728 - Why Projection is needed in parallel algorithms?
• PR #2727 - Cmake files for StackOverflow Detection
• PR #2726 - CMake for Stack Overflow Detection
• PR #2725 - Implemented segmented algorithms for partitioned vector
• PR #2724 - Fix examples in Action documentation
• PR #2723 - Enable lcos::channel<T>::register_as
• Issue #2722 - channel register_as() failing on compilation

https://github.com/STEllAR-GROUP/hpx/pull/2745
https://github.com/STEllAR-GROUP/hpx/pull/2744
https://github.com/STEllAR-GROUP/hpx/issues/2743
https://github.com/STEllAR-GROUP/hpx/pull/2742
https://github.com/STEllAR-GROUP/hpx/issues/2741
https://github.com/STEllAR-GROUP/hpx/pull/2740
https://github.com/STEllAR-GROUP/hpx/pull/2739
https://github.com/STEllAR-GROUP/hpx/pull/2738
https://github.com/STEllAR-GROUP/hpx/pull/2737
https://github.com/STEllAR-GROUP/hpx/issues/2736
https://github.com/STEllAR-GROUP/hpx/issues/2735
https://github.com/STEllAR-GROUP/hpx/pull/2734
https://github.com/STEllAR-GROUP/hpx/pull/2733
https://github.com/STEllAR-GROUP/hpx/pull/2732
https://github.com/STEllAR-GROUP/hpx/pull/2731
https://github.com/STEllAR-GROUP/hpx/issues/2729
https://github.com/STEllAR-GROUP/hpx/issues/2728
https://github.com/STEllAR-GROUP/hpx/pull/2727
https://github.com/STEllAR-GROUP/hpx/pull/2726
https://github.com/STEllAR-GROUP/hpx/pull/2725
https://github.com/STEllAR-GROUP/hpx/pull/2724
https://github.com/STEllAR-GROUP/hpx/pull/2723
https://github.com/STEllAR-GROUP/hpx/issues/2722
- PR #2721 - Mind map
- PR #2720 - reorder forward declarations to get rid of C++14-only auto return types
- PR #2719 - Add documentation for partitioned_vector and add features in pack.hpp
- Issue #2718 - Some forward declarations in execution_fwd.hpp aren’t C++11-compatible
- PR #2717 - Config support for fallthrough attribute
- PR #2716 - Implement parallel::partition_copy
- PR #2715 - initial import of icu string serializer
- PR #2714 - initial import of valarray serializer
- PR #2713 - Remove slashes before CMAKE_FILES_DIRECTORY variables
- PR #2712 - Fixing wait for 1751
- PR #2711 - Adjust code for minimal supported GCC having being bumped to 4.9
- PR #2710 - Adding code of conduct
- PR #2709 - Fixing UB in destroy tests
- PR #2708 - Add inline to prevent multiple definition issue
- Issue #2707 - Multiple defined symbols for task_block.hpp in VS2015
- PR #2706 - Adding .clang-format file
- PR #2704 - Add a synchronous mapping API
- Issue #2703 - Request: Add the .clang-format file to the repository
- Issue #2702 - STEllAR-GROUP/Vc slower than VCv1 possibly due to wrong instructions generated
- Issue #2701 - Datapar with STEllAR-GROUP/Vc requires obscure flag
- Issue #2700 - Naming inconsistency in parallel algorithms
- Issue #2699 - Iterator requirements are different from standard in parallel copy_if.
- PR #2698 - Properly releasing parcelport write handlers
<table>
<thead>
<tr>
<th>Issue #2697</th>
<th>Compile error in addressing_service.cpp</th>
</tr>
</thead>
<tbody>
<tr>
<td>Issue #2696</td>
<td>Building and using HPX statically: undefined references from runtime_support_server.cpp</td>
</tr>
<tr>
<td>Issue #2695</td>
<td>Executor changes cause compilation failures</td>
</tr>
<tr>
<td>PR #2694</td>
<td>Refining C++ language mode detection for MSVC</td>
</tr>
<tr>
<td>PR #2693</td>
<td>P0443 r2</td>
</tr>
<tr>
<td>PR #2692</td>
<td>Partially reverting changes to parcel_wait</td>
</tr>
<tr>
<td>Issue #2689</td>
<td>HPX build fails when HPX_WITH_CUDA is enabled</td>
</tr>
<tr>
<td>PR #2688</td>
<td>Make Cuda Clang builds pass</td>
</tr>
<tr>
<td>PR #2687</td>
<td>Add an is_tuple_like trait for sequenceable type detection</td>
</tr>
<tr>
<td>PR #2686</td>
<td>Allowing throttling scheduler to be used without idle backoff</td>
</tr>
<tr>
<td>PR #2685</td>
<td>Add support of std::array to hpx::util::tuple_size and tuple_element</td>
</tr>
<tr>
<td>PR #2684</td>
<td>Adding new statistics performance counters</td>
</tr>
<tr>
<td>PR #2683</td>
<td>Replace boost::exception_ptr with std::exception_ptr</td>
</tr>
<tr>
<td>Issue #2682</td>
<td>HPX does not compile with HPX_WITH_THREAD_MANAGER_IDLE_BACKOFF=OFF</td>
</tr>
<tr>
<td>PR #2681</td>
<td>Attempt to fix problem in managed_component_base</td>
</tr>
<tr>
<td>PR #2680</td>
<td>Fix bad size during archive creation</td>
</tr>
<tr>
<td>Issue #2679</td>
<td>Mismatch between size of archive and container</td>
</tr>
<tr>
<td>Issue #2678</td>
<td>In parallel algorithm, other tasks are executed to the end even if an exception occurs in any task.</td>
</tr>
<tr>
<td>PR #2677</td>
<td>Adding include check for std::addressof</td>
</tr>
<tr>
<td>PR #2676</td>
<td>Adding parallel::destroy and destroy_n</td>
</tr>
<tr>
<td>PR #2675</td>
<td>Making sure statistics counters work as expected</td>
</tr>
<tr>
<td>PR #2674</td>
<td>Turning assertions into exceptions</td>
</tr>
<tr>
<td>PR #2673</td>
<td>Inhibit direct conversion from future&lt;future&lt;T&gt;&gt; -&gt; future&lt;void&gt;</td>
</tr>
</tbody>
</table>

1689  https://github.com/STEllAR-GROUP/hpx/issues/2697
1690  https://github.com/STEllAR-GROUP/hpx/issues/2696
1691  https://github.com/STEllAR-GROUP/hpx/issues/2695
1692  https://github.com/STEllAR-GROUP/hpx/pull/2694
1693  https://github.com/STEllAR-GROUP/hpx/pull/2693
1694  https://github.com/STEllAR-GROUP/hpx/pull/2692
1695  https://github.com/STEllAR-GROUP/hpx/issues/2689
1696  https://github.com/STEllAR-GROUP/hpx/pull/2688
1697  https://github.com/STEllAR-GROUP/hpx/pull/2687
1698  https://github.com/STEllAR-GROUP/hpx/pull/2686
1699  https://github.com/STEllAR-GROUP/hpx/pull/2685
1700  https://github.com/STEllAR-GROUP/hpx/pull/2684
1701  https://github.com/STEllAR-GROUP/hpx/pull/2683
1702  https://github.com/STEllAR-GROUP/hpx/issues/2682
1703  https://github.com/STEllAR-GROUP/hpx/pull/2681
1704  https://github.com/STEllAR-GROUP/hpx/pull/2680
1705  https://github.com/STEllAR-GROUP/hpx/pull/2679
1706  https://github.com/STEllAR-GROUP/hpx/issues/2678
1707  https://github.com/STEllAR-GROUP/hpx/pull/2677
1708  https://github.com/STEllAR-GROUP/hpx/pull/2676
1709  https://github.com/STEllAR-GROUP/hpx/pull/2675
1710  https://github.com/STEllAR-GROUP/hpx/pull/2674
1711  https://github.com/STEllAR-GROUP/hpx/pull/2673
- PR #2672 - C++17 invoke forms
- PR #2671 - Adding uninitialized_value_construct and uninitialized_value_construct_n
- PR #2670 - Integrate spmd multidimensionnal views for partitioned_vectors
- PR #2669 - Adding uninitialized_default_construct and uninitialized_default_construct_n
- PR #2668 - Fixing documentation index
- Issue #2667 - Ambiguity of nested hpx::future<void>’s
- Issue #2666 - Statistics Performance counter is not working
- PR #2664 - Adding uninitialized_move and uninitialized_move_n
- Issue #2663 - Seg fault in managed_component::get_base_gid, possibly cause by util::reinitializable_static
- Issue #2662 - Crash in managed_component::get_base_gid due to problem with util::reinitializable_static
- PR #2665 - Hide the detail namespace in doxygen per default
- PR #2660 - Add documentation to hpx::util::unwrapped and hpx::util::unwrapped2
- PR #2659 - Improve integration with vcpkg
- PR #2658 - Uniﬁy access_data trait for use in both, serialization and de-serialization
- PR #2657 - Removing hpx::lcos::queue<T>
- PR #2656 - Reduce MAX_TERMINATED_THREADS default, improve memory use on manycore cpus
- PR #2655 - Maintenance for emulate-deleted macros
- PR #2654 - Implement parallel is_heap and is_heap_until
- PR #2653 - Drop support for VS2013
- PR #2652 - This patch makes sure that all parcels in a batch are properly handled
- PR #2649 - Update docs (Table 18) - move transform to end
- Issue #2647 - hpx::parcelset::detail::parcel_data::has_continuation_ is uninitialized
- Issue #2644 - Some .vcxproj in the HPX.sln fail to build

1712 https://github.com/STEllAR-GROUP/hpx/pull/2672
1713 https://github.com/STEllAR-GROUP/hpx/pull/2671
1714 https://github.com/STEllAR-GROUP/hpx/pull/2670
1715 https://github.com/STEllAR-GROUP/hpx/pull/2669
1716 https://github.com/STEllAR-GROUP/hpx/pull/2668
1717 https://github.com/STEllAR-GROUP/hpx/issues/2667
1718 https://github.com/STEllAR-GROUP/hpx/issues/2666
1719 https://github.com/STEllAR-GROUP/hpx/pull/2664
1720 https://github.com/STEllAR-GROUP/hpx/issues/2663
1721 https://github.com/STEllAR-GROUP/hpx/pull/2662
1722 https://github.com/STEllAR-GROUP/hpx/pull/2665
1723 https://github.com/STEllAR-GROUP/hpx/pull/2660
1724 https://github.com/STEllAR-GROUP/hpx/pull/2659
1725 https://github.com/STEllAR-GROUP/hpx/pull/2658
1726 https://github.com/STEllAR-GROUP/hpx/pull/2657
1727 https://github.com/STEllAR-GROUP/hpx/pull/2656
1728 https://github.com/STEllAR-GROUP/hpx/pull/2655
1729 https://github.com/STEllAR-GROUP/hpx/pull/2654
1730 https://github.com/STEllAR-GROUP/hpx/pull/2653
1731 https://github.com/STEllAR-GROUP/hpx/pull/2652
1732 https://github.com/STEllAR-GROUP/hpx/pull/2649
1733 https://github.com/STEllAR-GROUP/hpx/issues/2647
1734 https://github.com/STEllAR-GROUP/hpx/issues/2644
• Issue #2641 - hpx::lcos::queue should be deprecated
• PR #2640 - A new throttling policy with public APIs to suspend/resume
• PR #2639 - Fix a tiny typo in tutorial.
• Issue #2638 - Invalid return type ‘void’ of constexpr function
• PR #2636 - Add and use HPX_MSVC_WARNING_PRAGMA for #pragma warning
• PR #2634 - Distributed define_spmd_block
• PR #2632 - Making sure container serialization uses size-compatible types
• PR #2631 - Add lcos::local::one_element_channel
• PR #2629 - Move unordered_map out of parcelport into hpx/concurrent
• PR #2628 - Making sure that shutdown does not hang
• PR #2627 - Fix serialization
• PR #2626 - Generate cmake_variables.qbk and cmake_toolchains.qbk outside of the source tree
• PR #2625 - Supporting -std=c++17 flag
• PR #2624 - Fixing a small cmake typo
• PR #2622 - Update CMake minimum required version to 3.0.2 (closes #2621)
• Issue #2621 - Compiling hpx master fails with /usr/bin/ld: final link failed: Bad value
• PR #2620 - Remove warnings due to some captured variables
• PR #2619 - LF multiple parcels
• PR #2618 - Some fixes to libfabric that didn’t get caught before the merge
• PR #2617 - Adding hpx::local::new
• PR #2616 - Documentation: Extract all entities in order to autolink functions correctly
• Issue #2615 - Documentation: Linking functions is broken
• PR #2614 - Adding serialization for std::deque

1735 https://github.com/STEllAR-GROUP/hpx/issues/2641
1736 https://github.com/STEllAR-GROUP/hpx/pull/2640
1737 https://github.com/STEllAR-GROUP/hpx/pull/2639
1738 https://github.com/STEllAR-GROUP/hpx/issues/2638
1739 https://github.com/STEllAR-GROUP/hpx/pull/2636
1740 https://github.com/STEllAR-GROUP/hpx/pull/2633
1741 https://github.com/STEllAR-GROUP/hpx/pull/2632
1742 https://github.com/STEllAR-GROUP/hpx/pull/2631
1743 https://github.com/STEllAR-GROUP/hpx/pull/2629
1744 https://github.com/STEllAR-GROUP/hpx/pull/2628
1745 https://github.com/STEllAR-GROUP/hpx/pull/2627
1746 https://github.com/STEllAR-GROUP/hpx/pull/2626
1747 https://github.com/STEllAR-GROUP/hpx/pull/2625
1748 https://github.com/STEllAR-GROUP/hpx/pull/2624
1749 https://github.com/STEllAR-GROUP/hpx/pull/2622
1750 https://github.com/STEllAR-GROUP/hpx/issues/2621
1751 https://github.com/STEllAR-GROUP/hpx/pull/2620
1752 https://github.com/STEllAR-GROUP/hpx/pull/2619
1753 https://github.com/STEllAR-GROUP/hpx/pull/2618
1754 https://github.com/STEllAR-GROUP/hpx/pull/2617
1755 https://github.com/STEllAR-GROUP/hpx/pull/2616
1756 https://github.com/STEllAR-GROUP/hpx/issues/2615
1757 https://github.com/STEllAR-GROUP/hpx/pull/2614
• PR #2613 - We need to link with boost.thread and boost.chrono if we use boost.context
• PR #2612 - Making sure for_loop_n(par, ...) is actually executed in parallel
• PR #2611 - Add documentation to invoke_fused and friends NFC
• PR #2610 - Added reduction templates using an identity value
• PR #2608 - Fixing some unused vars in inspect
• PR #2607 - Fixed build for mingw
• PR #2606 - Supporting generic context for boost >= 1.61
• PR #2605 - Parcelport libfabric3
• PR #2604 - Adding allocator support to promise and friends
• PR #2603 - Barrier hang
• PR #2602 - Changes to scheduler to steal from one high-priority queue
• Issue #2601 - High priority tasks are not executed first
• PR #2600 - Compat fixes
• PR #2599 - Compatibility layer for threading support
• PR #2598 - V1.1
• PR #2597 - Release V1.0
• PR #2592 - First attempt to introduce spmd_block in hpx
• PR #2586 - local_segment in segmented_iterator_traits
• Issue #2584 - Add allocator support to promise, packaged_task and friends
• PR #2576 - Add missing dependencies of cuda based tests
• PR #2575 - Remove warnings due to some captured variables
• Issue #2574 - MSVC 2015 Compiler crash when building HPX
• Issue #2568 - Remove throttle_scheduler as it has been abandoned

1758 https://github.com/STEllAR-GROUP/hpx/pull/2613
1759 https://github.com/STEllAR-GROUP/hpx/pull/2612
1760 https://github.com/STEllAR-GROUP/hpx/pull/2611
1761 https://github.com/STEllAR-GROUP/hpx/pull/2610
1762 https://github.com/STEllAR-GROUP/hpx/pull/2608
1763 https://github.com/STEllAR-GROUP/hpx/pull/2607
1764 https://github.com/STEllAR-GROUP/hpx/pull/2606
1765 https://github.com/STEllAR-GROUP/hpx/pull/2605
1766 https://github.com/STEllAR-GROUP/hpx/pull/2604
1767 https://github.com/STEllAR-GROUP/hpx/pull/2603
1768 https://github.com/STEllAR-GROUP/hpx/pull/2602
1769 https://github.com/STEllAR-GROUP/hpx/issues/2601
1770 https://github.com/STEllAR-GROUP/hpx/pull/2600
1771 https://github.com/STEllAR-GROUP/hpx/pull/2599
1772 https://github.com/STEllAR-GROUP/hpx/pull/2598
1773 https://github.com/STEllAR-GROUP/hpx/pull/2597
1774 https://github.com/STEllAR-GROUP/hpx/pull/2592
1775 https://github.com/STEllAR-GROUP/hpx/pull/2586
1776 https://github.com/STEllAR-GROUP/hpx/issues/2584
1777 https://github.com/STEllAR-GROUP/hpx/pull/2576
1778 https://github.com/STEllAR-GROUP/hpx/pull/2575
1779 https://github.com/STEllAR-GROUP/hpx/issues/2574
1780 https://github.com/STEllAR-GROUP/hpx/issues/2568
• Issue #2566 - Add an inline versioning namespace before 1.0 release
• Issue #2565 - Raise minimal cmake version requirement
• PR #2556 - Fixing scan partitioner
• PR #2546 - Broadcast async
• Issue #2543 - make install fails due to a non-existing .so file
• PR #2495 - wait_or_add_new returning thread_id_type
• Issue #2480 - Unable to register new performance counter
• Issue #2471 - no type named ‘fcontext_t’ in namespace
• Issue #2456 - Re-implement hpx::util::unwrapped
• Issue #2455 - Add more arithmetic performance counters
• PR #2454 - Fix a couple of warnings and compiler errors
• PR #2453 - Timed executor support
• PR #2447 - Implementing new executor API (P0443)
• Issue #2439 - Implement executor proposal
• Issue #2408 - Stackoverflow detection for linux, e.g. based on libsigsegv
• PR #2377 - Add a customization point for put_parcel so we can override actions
• Issue #2368 - HPX_ASSERT problem
• Issue #2324 - Change default number of threads used to the maximum of the system
• Issue #2266 - hpx_0.9.99 make tests fail
• PR #2195 - Support for code completion in VIM
• Issue #2137 - Hpx does not compile over osx
• Issue #2092 - make tests should just build the tests
• Issue #2026 - Build HPX with Apple’s clang

1781 https://github.com/STEllAR-GROUP/hpx/issues/2566
1782 https://github.com/STEllAR-GROUP/hpx/issues/2565
1783 https://github.com/STEllAR-GROUP/hpx/pull/2556
1784 https://github.com/STEllAR-GROUP/hpx/pull/2546
1785 https://github.com/STEllAR-GROUP/hpx/issues/2543
1786 https://github.com/STEllAR-GROUP/hpx/pull/2495
1787 https://github.com/STEllAR-GROUP/hpx/issues/2480
1788 https://github.com/STEllAR-GROUP/hpx/issues/2471
1789 https://github.com/STEllAR-GROUP/hpx/issues/2456
1790 https://github.com/STEllAR-GROUP/hpx/issues/2455
1791 https://github.com/STEllAR-GROUP/hpx/pull/2454
1792 https://github.com/STEllAR-GROUP/hpx/pull/2453
1793 https://github.com/STEllAR-GROUP/hpx/pull/2447
1794 https://github.com/STEllAR-GROUP/hpx/issues/2439
1795 https://github.com/STEllAR-GROUP/hpx/issues/2408
1796 https://github.com/STEllAR-GROUP/hpx/pull/2377
1797 https://github.com/STEllAR-GROUP/hpx/issues/2368
1798 https://github.com/STEllAR-GROUP/hpx/issues/2324
1799 https://github.com/STEllAR-GROUP/hpx/issues/2266
1800 https://github.com/STEllAR-GROUP/hpx/pull/2195
1801 https://github.com/STEllAR-GROUP/hpx/issues/2137
1802 https://github.com/STEllAR-GROUP/hpx/pull/2092
1803 https://github.com/STEllAR-GROUP/hpx/issues/2026
• Issue #1932 - hpx with PBS fails on multiple localities
• PR #1914 - Parallel heap algorithm implementations WIP
• Issue #1598 - Disconnecting a locality results in segfault using heartbeat example
• Issue #1404 - unwrapped doesn’t work with movable only types
• Issue #1400 - hpx::util::unwrapped doesn’t work with non-future types
• Issue #1205 - TSS is broken
• Issue #1126 - vector<future<T>> does not work gracefully with dataflow, when_all and unwrapped
• Issue #1056 - Thread manager cleanup
• Issue #863 - Futures should not require a default constructor
• Issue #856 - Allow runtimemode_connect to be used with security enabled
• Issue #726 - Valgrind
• Issue #701 - Add RCR performance counter component
• Issue #528 - Add support for known failures and warning count/comparisons to hpx_run_tests.py

2.11.7 HPX V1.0.0 (Apr 24, 2017)

General changes
Here are some of the main highlights and changes for this release (in no particular order):

• Added the facility hpx::split_future which allows to convert a future<tuple<Ts...>> into a
  tuple<future<Ts>...>. This functionality is not available when compiling HPX with VS2012.

• Added a new type of performance counter which allows to return a list of values for each invocation. We also
  added a first counter of this type which collects a histogram of the times between parcels being created.

• Added new LCOs: hpx::lcos::channel and hpx::lcos::local::channel which are very similar
  to the well known channel constructs used in the Go language.

• Added new performance counters reporting the amount of data handled by the networking layer on a
  action-by-action basis (please see PR #2289 for more details).

• Added a new facility hpx::lcos::barrier, replacing the equally named older one. The new facility has
  a slightly changed API and is much more efficient. Most notable, the new facility exposes a (global) function
  hpx::lcos::barrier::synchronize() which represents a global barrier across all localities.

• We have started to add support for vectorization to our parallel algorithm implementations. This support depends
  on using an external library, currently either Vc Library or boost_simd. Please see Issue #2333 for a list

[1804] https://github.com/STEllAR-GROUP/hpx/issues/1932
[1805] https://github.com/STEllAR-GROUP/hpx/pull/1914
[1806] https://github.com/STEllAR-GROUP/hpx/issues/1598
[1807] https://github.com/STEllAR-GROUP/hpx/issues/1404
[1808] https://github.com/STEllAR-GROUP/hpx/issues/1400
[1809] https://github.com/STEllAR-GROUP/hpx/issues/1205
[1810] https://github.com/STEllAR-GROUP/hpx/issues/1126
[1811] https://github.com/STEllAR-GROUP/hpx/issues/1056
[1812] https://github.com/STEllAR-GROUP/hpx/issues/863
[1813] https://github.com/STEllAR-GROUP/hpx/issues/856
[1814] https://github.com/STEllAR-GROUP/hpx/issues/726
[1815] https://github.com/STEllAR-GROUP/hpx/issues/701
[1816] https://github.com/STEllAR-GROUP/hpx/issues/528
[1817] https://github.com/STEllAR-GROUP/hpx/pull/2289
[1818] https://github.com/STEllAR-GROUP/hpx/issues/2333

2.11. Releases
of currently supported algorithms. This is an experimental feature and its implementation and/or API might change in the future. Please see this blog-post\textsuperscript{1819} for more information.

- The parameter sequence for the `hpx::parallel::transform_reduce` overload taking one iterator range has changed to match the changes this algorithm has undergone while being moved to C++17. The old overload can be still enabled at configure time by specifying `-DHPX_WITH_TRANSFORM_REDUCE_COMPATIBILITY=On` to CMake.

- The algorithm `hpx::parallel::inner_product` has been renamed to `hpx::parallel::transform_reduce` to match the changes this algorithm has undergone while being moved to C++17. The old inner_product names can be still enabled at configure time by specifying `-DHPX_WITH_TRANSFORM_REDUCE_COMPATIBILITY=On` to CMake.

- Added versions of `hpx::get_ptr` taking client side representations for component instances as their parameter (instead of a global id).

- Added the helper utility `hpx::performance_counters::performance_counter_set` helping to encapsulate a set of performance counters to be managed concurrently.

- All execution policies and related classes have been renamed to be consistent with the naming changes applied for C++17. All policies now live in the namespace `hpx::parallel::execution`. The ols names can be still enabled at configure time by specifying `-DHPX_WITH_EXECUTION_POLICY_COMPATIBILITY=On` to CMake.

- The thread scheduling subsystem has undergone a major refactoring which results in significant performance improvements. We have also improved the performance of creating `hpx::future` and of various facilities handling those.

- We have consolidated all of the code in HPX.Compute related to the integration of CUDA. `hpx::partitioned_vector` has been enabled to be usable with `hpx::compute::vector` which allows to place the partitions on one or more GPU devices.

- Added new performance counters exposing various internals of the thread scheduling subsystem, such as the current idle- and busy-loop counters and instantaneous scheduler utilization.

- Extended and improved the use of the ITTNotify hooks allowing to collect performance counter data and function annotation information from within the Intel Amplifier tool.

### Breaking changes

- We have dropped support for the gcc compiler versions V4.6 and 4.7. The minimal gcc version we now test on is gcc V4.8.

- We have removed (default) support for `boost::chrono` in interfaces, uses of it have been replaced with `std::chrono`. This facility can be still enabled at configure time by specifying `-DHPX_WITH_BOOST_CHRONO_COMPATIBILITY=On` to CMake.

- The parameter sequence for the `hpx::parallel::transform_reduce` overload taking one iterator range has changed to match the changes this algorithm has undergone while being moved to C++17.

- The algorithm `hpx::parallel::inner_product` has been renamed to `hpx::parallel::transform_reduce` to match the changes this algorithm has undergone while being moved to C++17.

- the build options `HPX_WITH_COLOCATED_BACKWARDS_COMPATIBILITY` and `HPX_WITH_COMPONENT_GET_GID_COMPATIBILITY` are now disabled by default. Please change your code still depending on the deprecated interfaces.

\textsuperscript{1819} http://stellar-group.org/2016/09/vectorized-cpp-parallel-algorithms-with-hpx/
Bug fixes (closed tickets)

Here is a list of the important tickets we closed for this release.

- PR #2596[^1820] - Adding apex data
- PR #2595[^1821] - Remove obsolete file
- Issue #2594[^1822] - FindOpenCL.cmake mismatch with the official cmake module
- PR #2592[^1823] - First attempt to introduce spmd_block in hpx
- Issue #2591[^1824] - Feature request: continuation (then) which does not require the callable object to take a future<R> as parameter
- PR #2588[^1825] - Daint fixes
- PR #2587[^1826] - Fixing transfer_(continuation)_action::schedule
- PR #2585[^1827] - Work around MSVC having an ICE when compiling with -Ob2
- PR #2583[^1828] - chaning 7zip command to 7za in roll_release.sh
- PR #2582[^1829] - First attempt to introduce spmd_block in hpx
- PR #2581[^1830] - Enable annotated function for parallel algorithms
- PR #2580[^1831] - First attempt to introduce spmd_block in hpx
- PR #2579[^1832] - Make thread NICE level setting an option
- PR #2578[^1833] - Implementing enqueue instead of busy wait when no sender is available
- PR #2577[^1834] - Retrieve -std=c++11 consistent nvcc flag
- PR #2576[^1835] - Add missing dependencies of cuda based tests
- PR #2575[^1836] - Remove warnings due to some captured variables
- PR #2573[^1837] - Attempt to resolve resolve_locality
- PR #2572[^1838] - Adding APEX hooks to background thread
- PR #2571[^1839] - Pick up hpx.ignore_batch_env from config map
- PR #2570[^1840] - Add commandline options --hpx:print-counters-locally

[^1820]: https://github.com/STEllAR-GROUP/hpx/pull/2596
[^1821]: https://github.com/STEllAR-GROUP/hpx/pull/2595
[^1822]: https://github.com/STEllAR-GROUP/hpx/issues/2594
[^1823]: https://github.com/STEllAR-GROUP/hpx/pull/2592
[^1824]: https://github.com/STEllAR-GROUP/hpx/issues/2591
[^1825]: https://github.com/STEllAR-GROUP/hpx/pull/2588
[^1826]: https://github.com/STEllAR-GROUP/hpx/pull/2587
[^1827]: https://github.com/STEllAR-GROUP/hpx/pull/2585
[^1828]: https://github.com/STEllAR-GROUP/hpx/pull/2583
[^1829]: https://github.com/STEllAR-GROUP/hpx/pull/2582
[^1830]: https://github.com/STEllAR-GROUP/hpx/pull/2581
[^1831]: https://github.com/STEllAR-GROUP/hpx/pull/2580
[^1832]: https://github.com/STEllAR-GROUP/hpx/pull/2579
[^1833]: https://github.com/STEllAR-GROUP/hpx/pull/2578
[^1834]: https://github.com/STEllAR-GROUP/hpx/pull/2577
[^1835]: https://github.com/STEllAR-GROUP/hpx/pull/2576
[^1836]: https://github.com/STEllAR-GROUP/hpx/pull/2575
[^1837]: https://github.com/STEllAR-GROUP/hpx/pull/2573
[^1838]: https://github.com/STEllAR-GROUP/hpx/pull/2572
[^1839]: https://github.com/STEllAR-GROUP/hpx/pull/2571
[^1840]: https://github.com/STEllAR-GROUP/hpx/pull/2570
• PR #2569 - Fix computeapi unit tests
• PR #2567 - This adds another barrier::synchronize before registering performance counters
• PR #2564 - Cray static toolchain support
• PR #2563 - Fixed unhandled exception during startup
• PR #2562 - Remove partitioned_vector.cu from build tree when nvcc is used
• Issue #2561 - Octo-tiger crash with commit 6e921495ff6c26f125d62629cb40525f14f7ab
• PR #2560 - Prevent -Wundef warnings on Vc version checks
• PR #2559 - Allowing CUDA callback to set the future directly from an OS thread
• PR #2558 - Remove warnings due to float precisions
• PR #2557 - Removing bogus handling of compile flags for CUDA
• PR #2556 - Fixing scan partitioner
• PR #2554 - Add more diagnostics to error thrown from find_appropriate_destination
• Issue #2555 - No valid parcelport configured
• PR #2553 - Add cmake cuda_arch option
• PR #2552 - Remove incomplete datapar bindings to libflataarray
• PR #2551 - Rename hwloc_topolgy to hwloc_topolgy_info
• PR #2550 - Apex api updates
• PR #2549 - Pre-include defines.hpp to get the macro HPX_HAVE_CUAD value
• PR #2548 - Fixing issue with disconnect
• PR #2546 - Some fixes around cuda clang partitioned_vector example
• PR #2545 - Fix uses of the Vc2 datapar flags; the value, not the type, should be passed to functions
• PR #2542 - Make HPX_WITH_MALLOC easier to use
• PR #2541 - Avoid recompiles when enabling/disabling examples

https://github.com/STEllAR-GROUP/hpx/pull/2569
https://github.com/STEllAR-GROUP/hpx/pull/2567
https://github.com/STEllAR-GROUP/hpx/pull/2564
https://github.com/STEllAR-GROUP/hpx/pull/2563
https://github.com/STEllAR-GROUP/hpx/pull/2562
https://github.com/STEllAR-GROUP/hpx/issues/2561
https://github.com/STEllAR-GROUP/hpx/pull/2560
https://github.com/STEllAR-GROUP/hpx/pull/2559
https://github.com/STEllAR-GROUP/hpx/pull/2558
https://github.com/STEllAR-GROUP/hpx/pull/2557
https://github.com/STEllAR-GROUP/hpx/pull/2556
https://github.com/STEllAR-GROUP/hpx/pull/2554
https://github.com/STEllAR-GROUP/hpx/pull/2553
https://github.com/STEllAR-GROUP/hpx/pull/2552
https://github.com/STEllAR-GROUP/hpx/pull/2551
https://github.com/STEllAR-GROUP/hpx/pull/2550
https://github.com/STEllAR-GROUP/hpx/pull/2549
https://github.com/STEllAR-GROUP/hpx/pull/2548
https://github.com/STEllAR-GROUP/hpx/pull/2546
https://github.com/STEllAR-GROUP/hpx/pull/2545
https://github.com/STEllAR-GROUP/hpx/pull/2542
https://github.com/STEllAR-GROUP/hpx/pull/2541
• PR #2540 - Fixing usage of target_link_libraries()
• PR #2539 - Fix RPATH behaviour
• Issue #2538 - HPX_WITH_CUDA corrupts compilation flags
• PR #2537 - Add output of a Bazel Skylark extension for paths and compile options
• PR #2536 - Add counter exposing total available memory to Windows as well
• PR #2535 - Remove obsolete support for security
• Issue #2534 - Remove command line option --hpx:run-agas-server
• PR #2533 - Pre-cache locality endpoints during bootstrap
• PR #2532 - Fixing handling of GIDs during serialization preprocessing
• PR #2531 - Amend uses of the term “functor”
• PR #2530 - added counter for reading available memory
• PR #2529 - Facilities to create actions from lambdas
• PR #2528 - Updated docs: HPX_WITH_EXAMPLES
• PR #2527 - Remove warnings related to unused captured variables
• Issue #2526 - CMAKE failed because it is missing: TCMALLOC_LIBRARY TCMALLOC_INCLUDE_DIR
• PR #2525 - Fixing compose_cb stack overflow
• PR #2524 - Instead of unlocking, ignore the lock while creating the message handler
• PR #2523 - Create LPROGRESS logging macro to simplify progress tracking and timings
• PR #2522 - Intel 17 support
• PR #2521 - Fix components example
• PR #2520 - Fixing parcel scheduling
• Issue #2519 - Race condition during Parcel Coalescing Handler creation
• Issue #2518 - HPX locks up when using at least 256 localities

1864 https://github.com/STEllAR-GROUP/hpx/pull/2540
1865 https://github.com/STEllAR-GROUP/hpx/pull/2539
1866 https://github.com/STEllAR-GROUP/hpx/issues/2538
1867 https://github.com/STEllAR-GROUP/hpx/pull/2537
1868 https://github.com/STEllAR-GROUP/hpx/pull/2536
1869 https://github.com/STEllAR-GROUP/hpx/pull/2535
1870 https://github.com/STEllAR-GROUP/hpx/issues/2534
1871 https://github.com/STEllAR-GROUP/hpx/pull/2533
1872 https://github.com/STEllAR-GROUP/hpx/pull/2532
1873 https://github.com/STEllAR-GROUP/hpx/pull/2531
1874 https://github.com/STEllAR-GROUP/hpx/pull/2529
1875 https://github.com/STEllAR-GROUP/hpx/pull/2527
1876 https://github.com/STEllAR-GROUP/hpx/pull/2526
1877 https://github.com/STEllAR-GROUP/hpx/pull/2525
1878 https://github.com/STEllAR-GROUP/hpx/issues/2524
1879 https://github.com/STEllAR-GROUP/hpx/pull/2523
1880 https://github.com/STEllAR-GROUP/hpx/pull/2522
1881 https://github.com/STEllAR-GROUP/hpx/pull/2521
1882 https://github.com/STEllAR-GROUP/hpx/pull/2520
1883 https://github.com/STEllAR-GROUP/hpx/pull/2519
1884 https://github.com/STEllAR-GROUP/hpx/pull/2518
1885 https://github.com/STEllAR-GROUP/hpx/issues/2517
1886 https://github.com/STEllAR-GROUP/hpx/issues/2516
• Issue #2515 - error: Install cannot find “/lib/hpx/libparcel_coalescing.so.0.9.99” but I can see that file
• PR #2514 - Making sure that all continuations of a shared_future are invoked in order
• PR #2513 - Fixing locks held during suspension
• PR #2512 - MPI Parcelport improvements and fixes related to the background work changes
• PR #2511 - Fixing bit-wise (zero-copy) serialization
• Issue #2509 - Linking errors in hwloc_topology
• PR #2508 - Added documentation for debugging with core files
• PR #2506 - Fixing background work invocations
• PR #2505 - Fix tuple serialization
• Issue #2504 - Ensure continuations are called in the order they have been attached
• PR #2503 - Adding serialization support for Vc v2 (datapar)
• PR #2502 - Resolve various, minor compiler warnings
• PR #2501 - Some other fixes around cuda examples
• Issue #2500 - nvcc / cuda clang issue due to a missing -DHPX_WITH_CUDA flag
• PR #2499 - Adding support for std::array to wait_all and friends
• PR #2498 - Execute background work as HPX thread
• PR #2497 - Fixing configuration options for spinlock-deadlock detection
• PR #2496 - Accounting for different compilers in CrayKNL toolchain file
• PR #2494 - Adding component base class which ties a component instance to a given executor
• PR #2493 - Enable controlling amount of pending threads which must be available to allow thread stealing
• PR #2492 - Adding new command line option –hpx:print-counter-reset
• PR #2491 - Resolve ambiguities when compiling with APEX
• PR #2490 - Resuming threads waiting on future with higher priority

https://github.com/STEllAR-GROUP/hpx/issues/2515
https://github.com/STEllAR-GROUP/hpx/pull/2514
https://github.com/STEllAR-GROUP/hpx/pull/2513
https://github.com/STEllAR-GROUP/hpx/pull/2512
https://github.com/STEllAR-GROUP/hpx/pull/2511
https://github.com/STEllAR-GROUP/hpx/issues/2509
https://github.com/STEllAR-GROUP/hpx/pull/2508
https://github.com/STEllAR-GROUP/hpx/pull/2506
https://github.com/STEllAR-GROUP/hpx/pull/2505
https://github.com/STEllAR-GROUP/hpx/pull/2504
https://github.com/STEllAR-GROUP/hpx/pull/2503
https://github.com/STEllAR-GROUP/hpx/pull/2502
https://github.com/STEllAR-GROUP/hpx/pull/2501
https://github.com/STEllAR-GROUP/hpx/issues/2500
https://github.com/STEllAR-GROUP/hpx/pull/2499
https://github.com/STEllAR-GROUP/hpx/pull/2498
https://github.com/STEllAR-GROUP/hpx/pull/2497
https://github.com/STEllAR-GROUP/hpx/pull/2496
https://github.com/STEllAR-GROUP/hpx/pull/2494
https://github.com/STEllAR-GROUP/hpx/pull/2493
https://github.com/STEllAR-GROUP/hpx/pull/2492
https://github.com/STEllAR-GROUP/hpx/pull/2491
https://github.com/STEllAR-GROUP/hpx/pull/2490
• Issue #2489\textsuperscript{1910} - nvcc issue because -std=c++11 appears twice
• PR #2488\textsuperscript{1911} - Adding performance counters exposing the internal idle and busy-loop counters
• PR #2487\textsuperscript{1912} - Allowing for plain suspend to reschedule thread right away
• PR #2486\textsuperscript{1913} - Only flag HPX code for CUDA if HPX\_WITH\_CUDA is set
• PR #2485\textsuperscript{1914} - Making thread-queue parameters runtime-configurable
• PR #2484\textsuperscript{1915} - Added atomic counter for parcel-destinations
• PR #2483\textsuperscript{1916} - Added priority-queue lifo scheduler
• PR #2482\textsuperscript{1917} - Changing scheduler to steal only if more than a minimal number of tasks are available
• PR #2481\textsuperscript{1918} - Extending command line option –hpx:print-counter-destination to support value ‘none’
• PR #2479\textsuperscript{1919} - Added option to disable signal handler
• PR #2478\textsuperscript{1920} - Making sure the sine performance counter module gets loaded only for the corresponding example
• Issue #2477\textsuperscript{1921} - Breaking at a throw statement
• PR #2476\textsuperscript{1922} - Annotated function
• PR #2475\textsuperscript{1923} - Ensure that using %osthread% during logging will not throw for non-hpx threads
• PR #2474\textsuperscript{1924} - Remove now superficial non\_direct actions from base\_lco and friends
• PR #2473\textsuperscript{1925} - Refining support for ITTNotify
• PR #2472\textsuperscript{1926} - Some fixes around hpx compute
• Issue #2470\textsuperscript{1927} - redefinition of boost::detail::spinlock
• Issue #2469\textsuperscript{1928} - Dataflow performance issue
• PR #2468\textsuperscript{1929} - Perf docs update
• PR #2466\textsuperscript{1930} - Guarantee to execute remote direct actions on HPX-thread
• PR #2465\textsuperscript{1931} - Improve demo : Async copy and fixed device handling
• PR #2464\textsuperscript{1932} - Adding performance counter exposing instantaneous scheduler utilization

\textsuperscript{1910}https://github.com/STEllAR-GROUP/hpx/issues/2489
\textsuperscript{1911}https://github.com/STEllAR-GROUP/hpx/pull/2488
\textsuperscript{1912}https://github.com/STEllAR-GROUP/hpx/pull/2487
\textsuperscript{1913}https://github.com/STEllAR-GROUP/hpx/pull/2486
\textsuperscript{1914}https://github.com/STEllAR-GROUP/hpx/pull/2485
\textsuperscript{1915}https://github.com/STEllAR-GROUP/hpx/pull/2484
\textsuperscript{1916}https://github.com/STEllAR-GROUP/hpx/pull/2483
\textsuperscript{1917}https://github.com/STEllAR-GROUP/hpx/pull/2482
\textsuperscript{1918}https://github.com/STEllAR-GROUP/hpx/pull/2481
\textsuperscript{1919}https://github.com/STEllAR-GROUP/hpx/pull/2479
\textsuperscript{1920}https://github.com/STEllAR-GROUP/hpx/pull/2478
\textsuperscript{1921}https://github.com/STEllAR-GROUP/hpx/issues/2477
\textsuperscript{1922}https://github.com/STEllAR-GROUP/hpx/pull/2476
\textsuperscript{1923}https://github.com/STEllAR-GROUP/hpx/pull/2475
\textsuperscript{1924}https://github.com/STEllAR-GROUP/hpx/pull/2474
\textsuperscript{1925}https://github.com/STEllAR-GROUP/hpx/pull/2473
\textsuperscript{1926}https://github.com/STEllAR-GROUP/hpx/pull/2472
\textsuperscript{1927}https://github.com/STEllAR-GROUP/hpx/issues/2470
\textsuperscript{1928}https://github.com/STEllAR-GROUP/hpx/issues/2469
\textsuperscript{1929}https://github.com/STEllAR-GROUP/hpx/pull/2468
\textsuperscript{1930}https://github.com/STEllAR-GROUP/hpx/pull/2466
\textsuperscript{1931}https://github.com/STEllAR-GROUP/hpx/pull/2465
\textsuperscript{1932}https://github.com/STEllAR-GROUP/hpx/pull/2464
• PR #2463\textsuperscript{1933} - Downcast to future<void>
• PR #2462\textsuperscript{1934} - Fixed usage of ITT-Notify API with Intel Amplifier
• PR #2461\textsuperscript{1935} - Cublas demo
• PR #2460\textsuperscript{1936} - Fixing thread bindings
• PR #2459\textsuperscript{1937} - Make -std=c++11 nvcc flag consistent for in-build and installed versions
• Issue #2457\textsuperscript{1938} - Segmentation fault when registering a partitioned vector
• PR #2452\textsuperscript{1939} - Properly releasing global barrier for unhandled exceptions
• PR #2451\textsuperscript{1940} - Fixing long shutdown times
• PR #2450\textsuperscript{1941} - Attempting to fix initialization errors on newer platforms (Boost V1.63)
• PR #2449\textsuperscript{1942} - Replace BOOST_COMPILER_FENCE with an HPX version
• PR #2448\textsuperscript{1943} - This fixes a possible race in the migration code
• PR #2445\textsuperscript{1944} - Fixing dataflow et.al. for futures or future-ranges wrapped into ref()
• PR #2444\textsuperscript{1945} - Fix segfaults
• PR #2443\textsuperscript{1946} - Issue 2442
  
  Issue #2442\textsuperscript{1947} - Mismatch between #if/#endif and namespace scope brackets in this_thread_executers.hpp
  Issue #2441\textsuperscript{1948} - undeclared identifier BOOST_COMPILER_FENCE
• PR #2440\textsuperscript{1949} - Knl build
• PR #2438\textsuperscript{1950} - Datapar backend
• PR #2437\textsuperscript{1951} - Adapt algorithm parameter sequence changes from C++17
• PR #2436\textsuperscript{1952} - Adapt execution policy name changes from C++17
• Issue #2435\textsuperscript{1953} - Trunk broken, undefined reference to hpx::thread::interrupt(hpx::thread::id, bool)
• PR #2434\textsuperscript{1954} - More fixes to resource manager
• PR #2433\textsuperscript{1955} - Added versions of hpx::get_ptr taking client side representations
• PR #2432\textsuperscript{1956} - Warning fixes
• PR #2431\textsuperscript{1957} - Adding facility representing set of performance counters
• PR #2430\textsuperscript{1958} - Fix parallel_executors thread spawning
• PR #2429\textsuperscript{1959} - Fix attribute warning for gcc
• Issue #2427\textsuperscript{1960} - Seg fault running octo-tiger with latest HPX commit
• Issue #2426\textsuperscript{1961} - Bug in 952f5c0bc29806f3fe0d5bc73f5356a7e027edeb causes immediate crash in Octo-tiger
• PR #2425\textsuperscript{1962} - Fix nvcc errors due to constexpr specifier
• Issue #2424\textsuperscript{1963} - Async action on component present on hpx::find_here is executing synchronously
• PR #2423\textsuperscript{1964} - Fix nvcc errors due to constexpr specifier
• PR #2422\textsuperscript{1965} - Implementing hpx::this_thread thread data functions
• PR #2421\textsuperscript{1966} - Adding benchmark for wait_all
• Issue #2420\textsuperscript{1967} - Returning object of a component client from another component action fails
• PR #2419\textsuperscript{1968} - Infiniband parcelport
• Issue #2418\textsuperscript{1969} - gcc + nvcc fails to compile code that uses partitioned_vector
• PR #2417\textsuperscript{1970} - Fixing context switching
• PR #2416\textsuperscript{1971} - Adding fixes and workarounds to allow compilation with nvcc/msvc (VS2015up3)
• PR #2415\textsuperscript{1972} - Fix errors coming from hpx compute examples
• PR #2414\textsuperscript{1973} - Fixing msvc12
• PR #2413\textsuperscript{1974} - Enable cuda/nvcc or cuda/clang when using add_hpx_executable()
• PR #2412\textsuperscript{1975} - Fix issue in HPX_SetupTarget.cmake when cuda is used
• PR #2411\textsuperscript{1976} - This fixes the core compilation issues with MSVC12
• Issue #2410\textsuperscript{1977} - undefined reference to opal_hwloc191_hwloc_
• PR #2409\textsuperscript{1978} - Fixing locking for channel and receive_buffer
- PR #2407 - Solving #2402 and #2403
- PR #2406 - Improve guards
- PR #2405 - Enable parallel::for_each for iterators returning proxy types
- PR #2404 - Forward the explicitly given result_type in the hpx invoke
- Issue #2403 - datarpar_execution + zip iterator: lambda arguments aren’t references
- Issue #2402 - datarpar algorithm instantiated with wrong type #2402
- PR #2401 - Added support for imported libraries to HPX_Libraries.cmake
- PR #2400 - Use CMake policy CMP0060
- Issue #2399 - Error trying to push back vector of futures to vector
- PR #2398 - Allow config #defines to be written out to custom config/defines.hpp
- Issue #2397 - CMMake generated config defines can cause tedious rebuilds category
- Issue #2396 - BOOST_ROOT paths are not used at link time
- PR #2395 - Fix target_link_libraries() issue when HPX Cuda is enabled
- Issue #2394 - Template compilation error using HPX_WITH_DATAPAR_LIBFLATARRAY
- PR #2393 - Fixing lock registration for recursive mutex
- PR #2392 - Add keywords in target_link_libraries in hpx_setup_target
- PR #2391 - Clang goroutines
- Issue #2390 - Adapt execution policy name changes from C++17
- PR #2389 - Chunk allocator and pool are not used and are obsolete
- PR #2388 - Adding functionalities to datarpar needed by octotiger
- PR #2387 - Fixing race condition for early parcels
- Issue #2386 - Lock registration broken for recursive_mutex
- PR #2385 - Datapar zip iterator

1979 https://github.com/STEllAR-GROUP/hpx/pull/2407
1980 https://github.com/STEllAR-GROUP/hpx/pull/2406
1981 https://github.com/STEllAR-GROUP/hpx/pull/2405
1982 https://github.com/STEllAR-GROUP/hpx/pull/2404
1983 https://github.com/STEllAR-GROUP/hpx/issues/2403
1984 https://github.com/STEllAR-GROUP/hpx/issues/2402
1985 https://github.com/STEllAR-GROUP/hpx/pull/2401
1986 https://github.com/STEllAR-GROUP/hpx/pull/2400
1987 https://github.com/STEllAR-GROUP/hpx/issues/2399
1988 https://github.com/STEllAR-GROUP/hpx/pull/2398
1989 https://github.com/STEllAR-GROUP/hpx/issues/2397
1990 https://github.com/STEllAR-GROUP/hpx/issues/2396
1991 https://github.com/STEllAR-GROUP/hpx/pull/2395
1993 https://github.com/STEllAR-GROUP/hpx/pull/2393
1994 https://github.com/STEllAR-GROUP/hpx/pull/2392
1995 https://github.com/STEllAR-GROUP/hpx/pull/2391
1996 https://github.com/STEllAR-GROUP/hpx/issues/2390
1997 https://github.com/STEllAR-GROUP/hpx/pull/2389
1998 https://github.com/STEllAR-GROUP/hpx/pull/2388
1999 https://github.com/STEllAR-GROUP/hpx/pull/2387
2000 https://github.com/STEllAR-GROUP/hpx/issues/2386
2001 https://github.com/STEllAR-GROUP/hpx/pull/2385
- PR #2384\textsuperscript{2002} - Fixing race condition in for_loop_reduction
- PR #2383\textsuperscript{2003} - Continuations
- PR #2382\textsuperscript{2004} - add LibFlatArray-based backend for datapar
- PR #2381\textsuperscript{2005} - remove unused typedef to get rid of compiler warnings
- PR #2380\textsuperscript{2006} - Tau cleanup
- PR #2379\textsuperscript{2007} - Can send immediate
- PR #2378\textsuperscript{2008} - Renaming copy_helper/copy_n_helper/move_helper/move_n_helper
- Issue #2376\textsuperscript{2009} - Boost trunk’s spinlock initializer fails to compile
- PR #2375\textsuperscript{2010} - Add support for minimal thread local data
- PR #2374\textsuperscript{2011} - Adding API functions set_config_entry_callback
- PR #2373\textsuperscript{2012} - Add a simple utility for debugging that gives suspended task backtraces
- PR #2372\textsuperscript{2013} - Barrier Fixes
- Issue #2370\textsuperscript{2014} - Can’t wait on a wrapped future
- PR #2369\textsuperscript{2015} - Fixing stable_partition
- PR #2368\textsuperscript{2016} - Fixing find_prefixes for Windows platforms
- PR #2367\textsuperscript{2017} - Testing for experimental/optional only in C++14 mode
- PR #2366\textsuperscript{2018} - Adding set_config_entry
- PR #2365\textsuperscript{2019} - Fix papi
- PR #2364\textsuperscript{2020} - Adding missing macros for new non-direct actions
- PR #2363\textsuperscript{2021} - Improve cmake output to help debug compiler incompatibility check
- PR #2362\textsuperscript{2022} - Fixing race condition in condition_variable
- PR #2361\textsuperscript{2023} - Fixing shutdown when parcels are still in flight
- Issue #2357\textsuperscript{2024} - failed to insert console_print_action into typename_to_id_t registry

\textsuperscript{2002} https://github.com/STEllAR-GROUP/hpx/pull/2384
\textsuperscript{2003} https://github.com/STEllAR-GROUP/hpx/pull/2383
\textsuperscript{2004} https://github.com/STEllAR-GROUP/hpx/pull/2382
\textsuperscript{2005} https://github.com/STEllAR-GROUP/hpx/pull/2381
\textsuperscript{2006} https://github.com/STEllAR-GROUP/hpx/pull/2380
\textsuperscript{2007} https://github.com/STEllAR-GROUP/hpx/pull/2379
\textsuperscript{2008} https://github.com/STEllAR-GROUP/hpx/pull/2378
\textsuperscript{2009} https://github.com/STEllAR-GROUP/hpx/issues/2376
\textsuperscript{2010} https://github.com/STEllAR-GROUP/hpx/pull/2375
\textsuperscript{2011} https://github.com/STEllAR-GROUP/hpx/pull/2374
\textsuperscript{2012} https://github.com/STEllAR-GROUP/hpx/pull/2373
\textsuperscript{2013} https://github.com/STEllAR-GROUP/hpx/pull/2372
\textsuperscript{2014} https://github.com/STEllAR-GROUP/hpx/issues/2370
\textsuperscript{2015} https://github.com/STEllAR-GROUP/hpx/pull/2369
\textsuperscript{2016} https://github.com/STEllAR-GROUP/hpx/pull/2367
\textsuperscript{2017} https://github.com/STEllAR-GROUP/hpx/pull/2366
\textsuperscript{2018} https://github.com/STEllAR-GROUP/hpx/pull/2364
\textsuperscript{2019} https://github.com/STEllAR-GROUP/hpx/pull/2363
\textsuperscript{2020} https://github.com/STEllAR-GROUP/hpx/pull/2362
\textsuperscript{2021} https://github.com/STEllAR-GROUP/hpx/pull/2361
\textsuperscript{2022} https://github.com/STEllAR-GROUP/hpx/pull/2360
\textsuperscript{2023} https://github.com/STEllAR-GROUP/hpx/pull/2359
\textsuperscript{2024} https://github.com/STEllAR-GROUP/hpx/issues/2357
- Fixing return type of get_iterator_tuple
- Fixing compilation against Boost 1.62
- Adding serialization for mask_type if CPU_COUNT > 64
- Adding hooks to tie in APEX into the parcel layer
- Compile errors when using intel 17 beta (for KNL) on edison
- Fix function vtable get_function_address implementation
- Build failure - master branch (4de09f5) with Intel Compiler v17
- Enabling zero-copy serialization support for std::vector>
- Adding test to verify #2334 is fixed
- Bug fixes for hpx.compute and hpx::lcos::channel
- Removing cmake “find” files that are in the APEX cmake Modules
- Implemented parallel::stable_partition
- Making hpx::lcos::channel usable with basename registration
- Fix a couple of examples that failed to compile after recent api changes
- Enabling APEX causes link errors
- Removing cmake “find” files that are in the APEX cmake Modules
- Implemented all existing datapar algorithms using Boost.SIMD
- Fixing 2338
- Possible race in sliding semaphore
- Adjust osu_latency test to measure window_size parcels in flight at once
- Allowing remote direct actions to be executed without spawning a task
- Making sure multiple components are properly initialized from arguments
- Cannot construct component with large vector on a remote locality

2025 https://github.com/STEllAR-GROUP/hpx/pull/2356
2026 https://github.com/STEllAR-GROUP/hpx/pull/2355
2027 https://github.com/STEllAR-GROUP/hpx/pull/2354
2028 https://github.com/STEllAR-GROUP/hpx/pull/2353
2029 https://github.com/STEllAR-GROUP/hpx/issues/2352
2030 https://github.com/STEllAR-GROUP/hpx/pull/2351
2031 https://github.com/STEllAR-GROUP/hpx/issues/2350
2032 https://github.com/STEllAR-GROUP/hpx/pull/2349
2033 https://github.com/STEllAR-GROUP/hpx/pull/2348
2034 https://github.com/STEllAR-GROUP/hpx/pull/2347
2035 https://github.com/STEllAR-GROUP/hpx/pull/2346
2036 https://github.com/STEllAR-GROUP/hpx/pull/2345
2037 https://github.com/STEllAR-GROUP/hpx/pull/2344
2038 https://github.com/STEllAR-GROUP/hpx/pull/2343
2039 https://github.com/STEllAR-GROUP/hpx/issues/2342
2040 https://github.com/STEllAR-GROUP/hpx/pull/2341
2041 https://github.com/STEllAR-GROUP/hpx/pull/2340
2042 https://github.com/STEllAR-GROUP/hpx/pull/2339
2043 https://github.com/STEllAR-GROUP/hpx/pull/2338
2044 https://github.com/STEllAR-GROUP/hpx/pull/2337
2045 https://github.com/STEllAR-GROUP/hpx/pull/2336
2046 https://github.com/STEllAR-GROUP/hpx/pull/2335
2047 https://github.com/STEllAR-GROUP/hpx/issues/2334
- PR #2332 - Fixing hpx::lcos::local::barrier
- PR #2331 - Updating APEX support to include OTF2
- PR #2330 - Support for data-parallelism for parallel algorithms
- Issue #2329 - Coordinate settings in cmake
- PR #2328 - fix LibGeoDecomp builds with HPX + GCC 5.3.0 + CUDA 8RC
- PR #2326 - Making scan_partitioner work (for now)
- Issue #2323 - Constructing a vector of components only correctly initializes the first component
- PR #2322 - Fix problems that bubbled up after merging #2278
- PR #2321 - Scalable barrier
- PR #2320 - Std flag fixes
- Issue #2319 - -std=c++14 and -std=c++1y with Intel can’t build recent Boost builds due to insufficient C++14 support; don’t enable these flags by default for Intel
- PR #2318 - Improve handling of –hpx:bind=<bind-spec>
- PR #2317 - Making sure command line warnings are printed once only
- PR #2316 - Fixing command line handling for default bind mode
- PR #2315 - Set_id_retrieved if set_id is present
- Issue #2314 - Warning for requested/allocated thread discrepancy is printed twice
- Issue #2313 - –hpx:print-bind doesn’t work with –hpx:pu-step
- Issue #2312 - –hpx:bind range specifier restrictions are overly restrictive
- Issue #2311 - hpx_0.9.99 out of project build fails
- PR #2310 - Simplify function registration
- PR #2309 - Spelling and grammar revisions in documentation (and some code)
- PR #2308 - Correct minor typo in the documentation
- PR #2305 - Cleaning up and fixing parcel coalescing

2048 https://github.com/STEllAR-GROUP/hpx/pull/2332
2049 https://github.com/STEllAR-GROUP/hpx/pull/2331
2050 https://github.com/STEllAR-GROUP/hpx/pull/2330
2051 https://github.com/STEllAR-GROUP/hpx/issues/2329
2052 https://github.com/STEllAR-GROUP/hpx/pull/2328
2053 https://github.com/STEllAR-GROUP/hpx/pull/2326
2054 https://github.com/STEllAR-GROUP/hpx/issues/2323
2055 https://github.com/STEllAR-GROUP/hpx/pull/2322
2056 https://github.com/STEllAR-GROUP/hpx/pull/2321
2057 https://github.com/STEllAR-GROUP/hpx/pull/2320
2058 https://github.com/STEllAR-GROUP/hpx/pull/2319
2059 https://github.com/STEllAR-GROUP/hpx/pull/2318
2060 https://github.com/STEllAR-GROUP/hpx/pull/2317
2061 https://github.com/STEllAR-GROUP/hpx/pull/2316
2062 https://github.com/STEllAR-GROUP/hpx/pull/2315
2063 https://github.com/STEllAR-GROUP/hpx/issues/2314
2064 https://github.com/STEllAR-GROUP/hpx/issues/2313
2065 https://github.com/STEllAR-GROUP/hpx/issues/2312
2066 https://github.com/STEllAR-GROUP/hpx/issues/2311
2067 https://github.com/STEllAR-GROUP/hpx/pull/2310
2068 https://github.com/STEllAR-GROUP/hpx/pull/2309
2069 https://github.com/STEllAR-GROUP/hpx/pull/2306
2070 https://github.com/STEllAR-GROUP/hpx/pull/2305

2.11. Releases
• PR #2304 - Inspect checks for stream related includes
• PR #2303 - Add functionality allowing to enumerate threads of given state
• PR #2301 - Algorithm overloads fix for VS2013
• PR #2300 - Use <cstdint>, add inspect checks
• PR #2299 - Replace boost::[c]ref with std::[c]ref, add inspect checks
• PR #2297 - Fixing compilation with no hw_loc
• PR #2296 - Hpx compute
• PR #2295 - Making sure for_loop(execution::par, 0, N, ...) is actually executed in parallel
• PR #2294 - Throwing exceptions if the runtime is not up and running
• PR #2293 - Removing unused parcel port code
• PR #2292 - Refactor function vtables
• PR #2291 - Fixing 2286
• PR #2290 - Simplify algorithm overloads
• PR #2289 - Adding performance counters reporting parcel related data on a per-action basis
• Issue #2288 - Remove dormant parcelports
• Issue #2286 - adjustments to parcel handling to support parcelports that do not need a connection cache
• PR #2285 - add CMake option to disable package export
• PR #2283 - Add more inspect checks for use of deprecated components
• Issue #2282 - Arithmetic exception in executor static chunker
• Issue #2281 - For loop doesn’t parallelize
• PR #2280 - Fixing 2277: build failure with PAPI
• PR #2279 - Child vs parent stealing
• Issue #2277 - master branch build failure (53c5b4f) with papi

2071 https://github.com/STEllAR-GROUP/hpx/pull/2304
2072 https://github.com/STEllAR-GROUP/hpx/pull/2303
2073 https://github.com/STEllAR-GROUP/hpx/pull/2301
2074 https://github.com/STEllAR-GROUP/hpx/pull/2300
2075 https://github.com/STEllAR-GROUP/hpx/pull/2299
2076 https://github.com/STEllAR-GROUP/hpx/pull/2297
2077 https://github.com/STEllAR-GROUP/hpx/pull/2296
2078 https://github.com/STEllAR-GROUP/hpx/pull/2295
2079 https://github.com/STEllAR-GROUP/hpx/pull/2294
2080 https://github.com/STEllAR-GROUP/hpx/pull/2293
2081 https://github.com/STEllAR-GROUP/hpx/pull/2292
2082 https://github.com/STEllAR-GROUP/hpx/pull/2291
2083 https://github.com/STEllAR-GROUP/hpx/pull/2290
2084 https://github.com/STEllAR-GROUP/hpx/pull/2289
2085 https://github.com/STEllAR-GROUP/hpx/issues/2288
2086 https://github.com/STEllAR-GROUP/hpx/issues/2286
2087 https://github.com/STEllAR-GROUP/hpx/pull/2285
2088 https://github.com/STEllAR-GROUP/hpx/pull/2283
2089 https://github.com/STEllAR-GROUP/hpx/issues/2282
2090 https://github.com/STEllAR-GROUP/hpx/issues/2281
2091 https://github.com/STEllAR-GROUP/hpx/pull/2280
2092 https://github.com/STEllAR-GROUP/hpx/pull/2279
2093 https://github.com/STEllAR-GROUP/hpx/issues/2277

Chapter 2. What’s so special about HPX?
• PR #2276 - Compile time launch policies
• PR #2275 - Replace boost::chrono with std::chrono in interfaces
• PR #2274 - Replace most uses of Boost.Assign with initializer list
• PR #2273 - Fixed typos
• PR #2272 - Inspect checks
• PR #2270 - Adding test verifying -lpix.os_threads=all
• PR #2269 - Added inspect check for now obsolete boost type traits
• PR #2268 - Moving more code into source files
• Issue #2267 - Add inspect support to deprecate Boost.TypeTraits
• PR #2266 - Adding channel LCO
• PR #2264 - Make support for std::ref mandatory
• PR #2263 - Constrain tuple_member forwarding constructor
• Issue #2262 - Test hpix.os_threads=all
• Issue #2261 - OS X: Error: no matching constructor for initialization of ‘hpix::lcos::local::condition_variable_any’
• Issue #2260 - Make support for std::ref mandatory
• PR #2259 - Remove most of Boost.MPL, Boost.EnableIf and Boost.TypeTraits
• PR #2258 - Fixing #2256
• PR #2257 - Fixing launch process
• Issue #2256 - Actions are not registered if not invoked
• PR #2255 - Coalescing histogram
• PR #2254 - Silence explicit initialization in copy-constructor warnings
• PR #2253 - Drop support for GCC 4.6 and 4.7
• PR #2252 - Prepare V1.0

2104 https://github.com/STEllAR-GROUP/hpx/pull/2276
2105 https://github.com/STEllAR-GROUP/hpx/pull/2275
2106 https://github.com/STEllAR-GROUP/hpx/pull/2274
2107 https://github.com/STEllAR-GROUP/hpx/pull/2273
2108 https://github.com/STEllAR-GROUP/hpx/pull/2272
2109 https://github.com/STEllAR-GROUP/hpx/pull/2271
2110 https://github.com/STEllAR-GROUP/hpx/pull/2270
2111 https://github.com/STEllAR-GROUP/hpx/pull/2269
2112 https://github.com/STEllAR-GROUP/hpx/pull/2268
2113 https://github.com/STEllAR-GROUP/hpx/issues/2267
2114 https://github.com/STEllAR-GROUP/hpx/pull/2265
2115 https://github.com/STEllAR-GROUP/hpx/pull/2264
2116 https://github.com/STEllAR-GROUP/hpx/pull/2263
2117 https://github.com/STEllAR-GROUP/hpx/pull/2262
2118 https://github.com/STEllAR-GROUP/hpx/issues/2261
2119 https://github.com/STEllAR-GROUP/hpx/pull/2260
2120 https://github.com/STEllAR-GROUP/hpx/pull/2259
2121 https://github.com/STEllAR-GROUP/hpx/pull/2258
2122 https://github.com/STEllAR-GROUP/hpx/pull/2257
2123 https://github.com/STEllAR-GROUP/hpx/issues/2256
2124 https://github.com/STEllAR-GROUP/hpx/pull/2255
2125 https://github.com/STEllAR-GROUP/hpx/pull/2254
2126 https://github.com/STEllAR-GROUP/hpx/pull/2253
2127 https://github.com/STEllAR-GROUP/hpx/pull/2252

2.11. Releases
• PR #2251 - Convert to 0.9.99
• PR #2249 - Adding iterator_facade and iterator_adaptor
• Issue #2248 - Need a feature to yield to a new task immediately
• PR #2246 - Adding split_future
• PR #2245 - Add an example for handing over a component instance to a dynamically launched locality
• Issue #2243 - Add example demonstrating AGAS symbolic name registration
• Issue #2242 - pkgconfig test broken on CentOS 7 / Boost 1.61
• Issue #2241 - Compilation error for partitioned vector in hpx_compute branch
• PR #2240 - Fixing termination detection on one locality
• Issue #2239 - Create a new facility lcos::split_all
• Issue #2236 - hpx::cout vs. std::cout
• PR #2232 - Implement local-only primary namespace service
• Issue #2147 - would like to know how much data is being routed by particular actions
• Issue #2109 - Warning while compiling hpx
• Issue #1973 - Setting INTERFACE_COMPILE_OPTIONS for hpx_init in CMake taints Fortran_FLAGS
• Issue #1864 - run_guarded using bound function ignores reference
• Issue #1754 - Running with TCP parcelport causes immediate crash or freeze
• Issue #1655 - Enable zip_iterator to be used with Boost traversal iterator categories
• Issue #1591 - Optimize AGAS for shared memory only operation
• Issue #1401 - Need an efficient infiniband parcelport
• Issue #1125 - Fix the IPC parcelport
• Issue #839 - Refactor ibverbs and shmem parcelport
• Issue #702 - Add instrumentation of parcel layer

https://github.com/STEllAR-GROUP/hpx/pull/2251
https://github.com/STEllAR-GROUP/hpx/pull/2249
https://github.com/STEllAR-GROUP/hpx/issues/2248
https://github.com/STEllAR-GROUP/hpx/pull/2246
https://github.com/STEllAR-GROUP/hpx/pull/2245
https://github.com/STEllAR-GROUP/hpx/issues/2243
https://github.com/STEllAR-GROUP/hpx/issues/2242
https://github.com/STEllAR-GROUP/hpx/issues/2241
https://github.com/STEllAR-GROUP/hpx/pull/2240
https://github.com/STEllAR-GROUP/hpx/issues/2239
https://github.com/STEllAR-GROUP/hpx/issues/2236
https://github.com/STEllAR-GROUP/hpx/pull/2232
https://github.com/STEllAR-GROUP/hpx/issues/2147
https://github.com/STEllAR-GROUP/hpx/issues/2109
https://github.com/STEllAR-GROUP/hpx/issues/1973
https://github.com/STEllAR-GROUP/hpx/issues/1864
https://github.com/STEllAR-GROUP/hpx/issues/1754
https://github.com/STEllAR-GROUP/hpx/issues/1655
https://github.com/STEllAR-GROUP/hpx/issues/1591
https://github.com/STEllAR-GROUP/hpx/issues/1401
https://github.com/STEllAR-GROUP/hpx/issues/1125
https://github.com/STEllAR-GROUP/hpx/issues/839
https://github.com/STEllAR-GROUP/hpx/issues/702
• Issue #668 - Implement ispc task interface
• Issue #533 - Thread queue/deque internal parameters should be runtime configurable
• Issue #475 - Create a means of combining performance counters into querysets

2.11.8 HPX V0.9.99 (Jul 15, 2016)

General changes

As the version number of this release hints, we consider this release to be a preview for the upcoming HPX V1.0. All of the functionalities we set out to implement for V1.0 are in place; all of the features we wanted to have exposed are ready. We are very happy with the stability and performance of HPX and we would like to present this release to the community in order for us to gather broad feedback before releasing V1.0. We still expect for some minor details to change, but on the whole this release represents what we would like to have in a V1.0.

Overall, since the last release we have had almost 1600 commits while closing almost 400 tickets. These numbers reflect the incredible development activity we have seen over the last couple of months. We would like to express a big ‘Thank you!’ to all contributors and those who helped to make this release happen.

The most notable addition in terms of new functionality available with this release is the full implementation of object migration (i.e. the ability to transparently move HPX components to a different compute node). Additionally, this release of HPX cleans up many minor issues and some API inconsistencies.

Here are some of the main highlights and changes for this release (in no particular order):

• We have fixed a couple of issues in AGAS and the parcel layer which have caused hangs, segmentation faults at exit, and a slowdown of applications over time. Fixing those has significantly increased the overall stability and performance of distributed runs.

• We have started to add parallel algorithm overloads based on the C++ Extensions for Ranges (N4560) proposal. This also includes the addition of projections to the existing algorithms. Please see Issue #1668 for a list of algorithms which have been adapted to N4560.

• We have implemented index-based parallel for-loops based on a corresponding standardization proposal (P0075R1). Please see Issue #2016 for a list of available algorithms.

• We have started to add parallel algorithm overloads based on the C++ Extensions for Ranges (N4560) proposal. This also includes the addition of projections to the existing algorithms. Please see Issue #1668 for a list of algorithms which have been adapted to N4560.

• We have added implementations for more parallel algorithms as proposed for the upcoming C++ 17 Standard. See Issue #1141 for an overview of which algorithms are available by now.

• We have started to implement a new prototypical functionality with HPX.Compute which uniformly exposes some of the higher level APIs to heterogeneous architectures (currently CUDA). This functionality is an early preview and should not be considered stable. It may change considerably in the future.

• We have pervasively added (optional) executor arguments to all API functions which schedule new work. Executors are now used throughout the code base as the main means of executing tasks.

• Added hpx::make_future<R>(future<T> &&) allowing to convert a future of any type T into a future of any other type R, either based on default conversion rules of the embedded types or using a given explicit conversion function.

• We finally finished the implementation of transparent migration of components to another locality. It is now possible to trigger a migration operation without ‘stopping the world’ for the object to migrate. HPX will make

2140 https://github.com/STEllAR-GROUP/hpx/issues/668
2141 https://github.com/STEllAR-GROUP/hpx/issues/533
2142 https://github.com/STEllAR-GROUP/hpx/issues/475
2143 http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2015/n4560.pdf
2144 https://github.com/STEllAR-GROUP/hpx/issues/1668
2147 https://github.com/STEllAR-GROUP/hpx/issues/2016
2148 https://github.com/STEllAR-GROUP/hpx/issues/1141
sure that no work is being performed on an object before it is migrated and that all subsequently scheduled work for the migrated object will be transparently forwarded to the new locality. Please note that the global id of the migrated object does not change, thus the application will not have to be changed in any way to support this new functionality. Please note that this feature is currently considered experimental. See Issue #559 and PR #1966 for more details.

- The `hpx::dataflow` facility is now usable with actions. Similarly to `hpx::async`, actions can be specified as an explicit template argument (`hpx::dataflow<Action>(target, ...)`) or as the first argument (`hpx::dataflow(Action(), target, ...)`). We have also enabled the use of distribution policies as the target for dataflow invocations. Please see Issue #1265 and PR #1912 for more information.

- Adding overloads of `gather_here` and `gather_there` to accept the plain values of the data to gather (in addition to the existing overloads expecting futures).

- We have cleaned up and refactored large parts of the code base. This helped reducing compile and link times of `HPX` itself and also of applications depending on it. We have further decreased the dependency of `HPX` on the Boost libraries by replacing part of those with facilities available from the standard libraries.

- Wherever possible we have removed dependencies of our API on Boost by replacing those with the equivalent facility from the C++11 standard library.

- We have added new performance counters for parcel coalescing, file-IO, the AGAS cache, and overall scheduler time. Resetting performance counters has been overhauled and fixed.

- We have introduced a generic client type `hpx::components::client<>` and added support for using it with `hpx::async`. This removes the necessity to implement specific client types for every component type without losing type safety. This deemphasizes the need for using the low level `hpx::id_type` for referencing (possibly remote) component instances. The plan is to deprecate the direct use of `hpx::id_type` in user code in the future.

- We have added a special iterator which supports automatic prefetching of one or more arrays for speeding up loop-like code (see `hpx::parallel::util::make_prefetcher_context()`).

- We have extended the interfaces exposed from executors (as proposed by N4406) to accept an arbitrary number of arguments.

### Breaking changes

- In order to move the dataflow facility to namespace `hpx` we added a definition of `hpx::dataflow` which might create ambiguities in existing codes. The previous definition of this facility (`hpx::lcos::local::dataflow`) has been deprecated and is available only if the constant `~DHPX_WITH_LOCAL_DATAFLOW_COMPATIBILITY=On` to CMake is defined at configuration time. Please explicitly qualify all uses of the dataflow facility if you enable this compatibility setting and encounter ambiguities.

- The adaptation of the C++ Extensions for Ranges (N4560) proposal imposes some breaking changes related to the return types of some of the parallel algorithms. Please see Issue #1668 for a list of algorithms which have already been adapted.

- The facility `hpx::lcos::make_future_void()` has been replaced by `hpx::make_future<void>()`.

- We have removed support for Intel V13 and gcc 4.4.x.

---

2149 https://github.com/STEllAR-GROUP/hpx/issues/559
2150 https://github.com/STEllAR-GROUP/hpx/pull/1966
2151 https://github.com/STEllAR-GROUP/hpx/issues/1265
2152 https://github.com/STEllAR-GROUP/hpx/pull/1912
2154 https://www.cmake.org
2156 https://github.com/STEllAR-GROUP/hpx/issues/1668
• We have removed (default) support for the generic hpx::parallel::execution_policy because it was removed from the Parallelism TS (__cpp11_n4104__) while it was being added to the upcoming C++17 Standard. This facility can be still enabled at configure time by specifying -DHPX_WITH GENERIC_EXECUTION_POLICY=On to CMak

• Uses of boost::shared_ptr and related facilities have been replaced with std::shared_ptr and friends. Uses of boost::unique_lock, boost::lock_guard etc. have also been replaced by the equivalent (and equally named) tools available from the C++11 standard library.

• Facilities that used to expect an explicit boost::unique_lock now take an std::unique_lock. Additionally, condition_variable no longer aliases condition_variable_any; its interface now only works with std::unique_lock<local::mutex>.

• Uses of boost::function, boost::bind, boost::tuple have been replaced by the corresponding facilities in HPX (hpx::util::function, hpx::util::bind, and hpx::util::tuple, respectively).

Bug fixes (closed tickets)

Here is a list of the important tickets we closed for this release.

• PR #2250\textsuperscript{2157} - change default chunker of parallel executor to static one
• PR #2247\textsuperscript{2158} - HPX on ppc64le
• PR #2244\textsuperscript{2159} - Fixing MSVC problems
• PR #2238\textsuperscript{2160} - Fixing small typos
• PR #2237\textsuperscript{2161} - Fixing small typos
• PR #2234\textsuperscript{2162} - Fix broken add test macro when extra args are passed in
• PR #2231\textsuperscript{2163} - Fixing possible race during future awaiting in serialization
• PR #2230\textsuperscript{2164} - Fix stream nvcc
• PR #2229\textsuperscript{2165} - Fixed run_as_hpx_thread
• PR #2228\textsuperscript{2166} - On prefetching_test branch : adding prefetching_iterator and related tests used for prefetching containers within lambda functions
• PR #2227\textsuperscript{2167} - Support for HPXCL’s opencl::event
• PR #2226\textsuperscript{2168} - Preparing for release of V0.9.99
• PR #2225\textsuperscript{2169} - fix issue when compiling components with hpxcxx
• PR #2224\textsuperscript{2170} - Compute alloc fix

\textsuperscript{2157} https://github.com/STEllAR-GROUP/hpx/pull/2250
\textsuperscript{2158} https://github.com/STEllAR-GROUP/hpx/pull/2247
\textsuperscript{2159} https://github.com/STEllAR-GROUP/hpx/pull/2244
\textsuperscript{2160} https://github.com/STEllAR-GROUP/hpx/pull/2238
\textsuperscript{2161} https://github.com/STEllAR-GROUP/hpx/pull/2237
\textsuperscript{2162} https://github.com/STEllAR-GROUP/hpx/pull/2234
\textsuperscript{2163} https://github.com/STEllAR-GROUP/hpx/pull/2231
\textsuperscript{2164} https://github.com/STEllAR-GROUP/hpx/pull/2230
\textsuperscript{2165} https://github.com/STEllAR-GROUP/hpx/pull/2229
\textsuperscript{2166} https://github.com/STEllAR-GROUP/hpx/pull/2228
\textsuperscript{2167} https://github.com/STEllAR-GROUP/hpx/pull/2227
\textsuperscript{2168} https://github.com/STEllAR-GROUP/hpx/pull/2226
\textsuperscript{2169} https://github.com/STEllAR-GROUP/hpx/pull/2225
\textsuperscript{2170} https://github.com/STEllAR-GROUP/hpx/pull/2224

2.11. Releases
• PR #2223 - Simplify promise
• PR #2222 - Replace last uses of boost::function by util::function_nonser
• PR #2221 - Fix config tests
• PR #2220 - Fixing gcc 4.6 compilation issues
• PR #2219 - nullptr support for [unique_]function
• PR #2218 - Introducing clang tidy
• PR #2216 - Replace NULL with nullptr
• Issue #2214 - Let inspect flag use of NULL, suggest nullptr instead
• PR #2213 - Require support for nullptr
• PR #2212 - Properly find jemalloc through pkg-config
• PR #2211 - Disable a couple of warnings reported by Intel on Windows
• PR #2210 - Fixed host::block_allocator::bulk_construct
• PR #2209 - Started to clean up new sort algorithms, made things compile for sort_by_key
• PR #2208 - A couple of fixes that were exposed by a new sort algorithm
• PR #2207 - Adding missing includes in /hpx/include/serialization.hpp
• PR #2206 - Call package_action::get_future before package_action::apply
• PR #2205 - The indirect_packaged_task::operator() needs to be run on a HPX thread
• PR #2204 - Variadic executor parameters
• PR #2203 - Delay-initialize members of partitioned iterator
• PR #2202 - Added segmented fill for hpx::vector
• Issue #2201 - Null Thread id encountered on partitioned_vector
• PR #2200 - Fix hangs
• PR #2199 - Deprecating hpx/traits.hpp

2171 https://github.com/STEllAR-GROUP/hpx/pull/2223
2172 https://github.com/STEllAR-GROUP/hpx/pull/2222
2173 https://github.com/STEllAR-GROUP/hpx/pull/2221
2174 https://github.com/STEllAR-GROUP/hpx/pull/2220
2175 https://github.com/STEllAR-GROUP/hpx/pull/2219
2176 https://github.com/STEllAR-GROUP/hpx/pull/2218
2177 https://github.com/STEllAR-GROUP/hpx/pull/2216
2178 https://github.com/STEllAR-GROUP/hpx/issues/2214
2179 https://github.com/STEllAR-GROUP/hpx/pull/2213
2180 https://github.com/STEllAR-GROUP/hpx/pull/2212
2181 https://github.com/STEllAR-GROUP/hpx/pull/2211
2182 https://github.com/STEllAR-GROUP/hpx/pull/2210
2183 https://github.com/STEllAR-GROUP/hpx/pull/2209
2184 https://github.com/STEllAR-GROUP/hpx/pull/2208
2185 https://github.com/STEllAR-GROUP/hpx/pull/2207
2186 https://github.com/STEllAR-GROUP/hpx/pull/2206
2187 https://github.com/STEllAR-GROUP/hpx/pull/2205
2188 https://github.com/STEllAR-GROUP/hpx/pull/2204
2189 https://github.com/STEllAR-GROUP/hpx/pull/2203
2190 https://github.com/STEllAR-GROUP/hpx/pull/2202
2191 https://github.com/STEllAR-GROUP/hpx/issues/2201
2192 https://github.com/STEllAR-GROUP/hpx/pull/2200
2193 https://github.com/STEllAR-GROUP/hpx/pull/2199
• PR #2198 - Making explicit inclusion of external libraries into build
• PR #2197 - Fix typo in QT CMakeLists
• PR #2196 - Fixing a gcc warning about attributes being ignored
• PR #2194 - Fixing partitioned_vector_spmd_foreach example
• Issue #2193 - partitioned_vector_spmd_foreach seg faults
• PR #2192 - Support Boost.Thread v4
• PR #2191 - HPX.Compute prototype
• PR #2190 - Spawning operation on new thread if remaining stack space becomes too small
• PR #2189 - Adding callback taking index and future to when_each
• PR #2188 - Adding new example demonstrating receive_buffer
• PR #2187 - Mask 128-bit ints if CUDA is being used
• PR #2186 - Make startup & shutdown functions unique_function
• PR #2185 - Fixing logging output not to cause hang on shutdown
• PR #2184 - Allowing component clients as action return types
• Issue #2183 - Enabling logging output causes hang on shutdown
• Issue #2182 - 1d_stencil seg fault
• Issue #2181 - Setting small stack size does not change default
• PR #2180 - Changing default bind mode to balanced
• PR #2179 - adding prefetching_iterator and related tests used for prefetching containers within lambda functions
• PR #2177 - Fixing 2176
• Issue #2176 - Launch process test fails on OSX
• PR #2175 - Fix unbalanced config/warnings includes, add some new ones
• PR #2174 - Fix test categorization: regression not unit
• Issue #2172 - Different performance results
• Issue #2171 - “negative entry in reference count table” running octotiger on 32 nodes on queenbee
• Issue #2170 - Error while compiling on Mac + boost 1.60
• PR #2168 - Fixing problems with is_bitwise_serializable
• Issue #2167 - startup & shutdown function should accept unique_function
• Issue #2166 - Simple receive_buffer example
• PR #2165 - Fix wait all
• PR #2164 - Fix wait all
• PR #2163 - Fix some typos in config tests
• PR #2162 - Improve #includes
• PR #2160 - Add inspect check for missing #include <list>
• PR #2159 - Add missing finalize call to stop test hanging
• PR #2158 - Algo fixes
• PR #2157 - Stack check
• Issue #2156 - OSX reports stack space incorrectly (generic context coroutines)
• Issue #2155 - Race condition suspected in runtime
• PR #2154 - Replace boost::detail::atomic_count with the new util::atomic_count
• PR #2153 - Fix stack overflow on OSX
• PR #2152 - Define is_bitwise_serializable as is_trivially_copyable when available
• PR #2151 - Adding missing <cstring> for std::mem* functions
• Issue #2150 - Unable to use component clients as action return types
• PR #2149 - std::memmove copies bytes, use bytes*sizeof(type) when copying larger types
• PR #2146 - Adding customization point for parallel copy/move
- PR #2145 - Applying changes to address warnings issued by latest version of PVS Studio
- Issue #2148 - hpx::parallel::copy is broken after trivially copyable changes
- PR #2144 - Some minor tweaks to compute prototype
- PR #2143 - Added Boost version support information over OSX platform
- PR #2142 - Fixing memory leak in example
- PR #2141 - Add missing specializations in execution policies
- PR #2139 - This PR fixes a few problems reported by Clang’s Undefined Behavior sanitizer
- PR #2138 - Revert “Adding fedora docs”
- PR #2137 - Removed double semicolon
- PR #2136 - Add deprecated #include check for hpx_fwd.hpp
- PR #2134 - Resolved memory leak in stencil_8
- PR #2133 - Replace uses of boost pointer containers
- PR #2132 - Removing unused typedef
- PR #2131 - Add several include checks for std facilities
- PR #2130 - Fixing parcel compression, adding test
- PR #2129 - Fix invalid attribute warnings
- Issue #2128 - hpx::init seems to segfault
- PR #2127 - Making executor_traits N-nary
- PR #2126 - GCC 4.6 fails to deduce the correct type in lambda
- PR #2125 - Making parcel coalescing test actually test something
- Issue #2124 - Make a testcase for parcel compression
- Issue #2123 - hpx/hpx/runtime/applier_fwd.hpp - Multiple defined types
- Issue #2122 - Exception in primary_namespace::resolve_free_list

https://github.com/STEllAR-GROUP/hpx/pull/2145
https://github.com/STEllAR-GROUP/hpx/issues/2148
https://github.com/STEllAR-GROUP/hpx/pull/2144
https://github.com/STEllAR-GROUP/hpx/pull/2143
https://github.com/STEllAR-GROUP/hpx/pull/2142
https://github.com/STEllAR-GROUP/hpx/pull/2141
https://github.com/STEllAR-GROUP/hpx/pull/2139
https://github.com/STEllAR-GROUP/hpx/pull/2138
https://github.com/STEllAR-GROUP/hpx/pull/2136
https://github.com/STEllAR-GROUP/hpx/pull/2135
https://github.com/STEllAR-GROUP/hpx/pull/2134
https://github.com/STEllAR-GROUP/hpx/pull/2133
https://github.com/STEllAR-GROUP/hpx/pull/2132
https://github.com/STEllAR-GROUP/hpx/pull/2131
https://github.com/STEllAR-GROUP/hpx/pull/2130
https://github.com/STEllAR-GROUP/hpx/pull/2129
https://github.com/STEllAR-GROUP/hpx/issues/2128
https://github.com/STEllAR-GROUP/hpx/pull/2127
https://github.com/STEllAR-GROUP/hpx/pull/2126
https://github.com/STEllAR-GROUP/hpx/pull/2125
https://github.com/STEllAR-GROUP/hpx/issues/2124
https://github.com/STEllAR-GROUP/hpx/issues/2123
https://github.com/STEllAR-GROUP/hpx/issues/2122
• Issue #2121 - Possible memory leak in 1d_stencil_8
• PR #2120 - Fixing 2119
• Issue #2119 - reduce_by_key compilation problems
• Issue #2118 - Premature unwrapping of boost::ref'ed arguments
• PR #2117 - Added missing initializer on last constructor for thread_description
• PR #2116 - Use a lightweight bind implementation when no placeholders are given
• PR #2115 - Replace boost::shared_ptr with std::shared_ptr
• PR #2114 - Adding hook functions for executor_parameter_traits supporting timers
• Issue #2113 - Compilation error with gcc version 4.9.3 (MacPorts gcc49 4.9.3_0)
• PR #2112 - Replace uses of safe_bool with explicit operator bool
• Issue #2111 - Compilation error on QT example
• Issue #2110 - Compilation error when passing non-future argument to unwrapped continuation in dataflow
• Issue #2109 - Warning while compiling hpx
• Issue #2109 - Stack trace of last bug causing issues with octotiger
• Issue #2108 - Stack trace of last bug causing issues with octotiger
• PR #2107 - Making sure that a missing parcel_coalescing module does not cause startup exceptions
• PR #2106 - Drop using hpx_fwd.hpp
• Issue #2105 - coalescing plugin handler is not optional any more
• Issue #2104 - Make executor_traits N-nary
• Issue #2103 - Build error with octotiger and hpx commit e657426d
• PR #2102 - Combining thread data storage
• PR #2101 - Added repartition version of 1d stencil that uses any performance counter
• PR #2100 - Drop obsolete TR1 result_of protocol

https://github.com/STEllAR-GROUP/hpx/issues/2121
https://github.com/STEllAR-GROUP/hpx/pull/2120
https://github.com/STEllAR-GROUP/hpx/issues/2119
https://github.com/STEllAR-GROUP/hpx/issues/2118
https://github.com/STEllAR-GROUP/hpx/pull/2117
https://github.com/STEllAR-GROUP/hpx/pull/2116
https://github.com/STEllAR-GROUP/hpx/pull/2115
https://github.com/STEllAR-GROUP/hpx/pull/2114
https://github.com/STEllAR-GROUP/hpx/issues/2113
https://github.com/STEllAR-GROUP/hpx/pull/2112
https://github.com/STEllAR-GROUP/hpx/issues/2111
https://github.com/STEllAR-GROUP/hpx/issues/2110
https://github.com/STEllAR-GROUP/hpx/issues/2109
https://github.com/STEllAR-GROUP/hpx/issues/2108
https://github.com/STEllAR-GROUP/hpx/pull/2107
https://github.com/STEllAR-GROUP/hpx/pull/2106
https://github.com/STEllAR-GROUP/hpx/issues/2105
https://github.com/STEllAR-GROUP/hpx/issues/2104
https://github.com/STEllAR-GROUP/hpx/issues/2103
https://github.com/STEllAR-GROUP/hpx/pull/2102
https://github.com/STEllAR-GROUP/hpx/pull/2101
https://github.com/STEllAR-GROUP/hpx/pull/2100
- PR #2099\textsuperscript{2286} - Replace uses of boost::bind with util::bind
- PR #2098\textsuperscript{2287} - Deprecated inspect checks
- PR #2097\textsuperscript{2288} - Reduce by key, extends #1141
- PR #2096\textsuperscript{2289} - Moving local cache from external to hpx/util
- PR #2095\textsuperscript{2290} - Bump minimum required Boost to 1.50.0
- PR #2094\textsuperscript{2291} - Add include checks for several Boost utilities
- Issue #2093\textsuperscript{2292} - /. . . /local_cache.hpp(89): error #303: explicit type is missing (“int” assumed)
- PR #2091\textsuperscript{2293} - Fix for Raspberry pi build
- PR #2090\textsuperscript{2294} - Fix storage size for util::function<>
- PR #2089\textsuperscript{2295} - Fix #2088
- Issue #2088\textsuperscript{2296} - More verbose output from cmake configuration
- PR #2087\textsuperscript{2297} - Making sure init_globally always executes hpx_main
- Issue #2086\textsuperscript{2298} - Race condition with recent HPX
- PR #2085\textsuperscript{2299} - Adding #include checker
- PR #2084\textsuperscript{2300} - Replace boost lock types with standard library ones
- PR #2083\textsuperscript{2301} - Simplify packaged task
- PR #2082\textsuperscript{2302} - Updating APEX version for testing
- PR #2081\textsuperscript{2303} - Cleanup exception headers
- PR #2080\textsuperscript{2304} - Make call_once variadic
- Issue #2079\textsuperscript{2305} - With GNU C++, line 85 of hpx/config/version.hpp causes link failure when linking application
- Issue #2078\textsuperscript{2306} - Simple test fails with _GLIBCXX_DEBUG defined
- PR #2077\textsuperscript{2307} - Instantiate board in nqueen client
- PR #2076\textsuperscript{2308} - Moving coalescing registration to TUs

\textsuperscript{2286} https://github.com/STEllAR-GROUP/hpx/pull/2099
\textsuperscript{2287} https://github.com/STEllAR-GROUP/hpx/pull/2098
\textsuperscript{2288} https://github.com/STEllAR-GROUP/hpx/pull/2097
\textsuperscript{2289} https://github.com/STEllAR-GROUP/hpx/pull/2096
\textsuperscript{2290} https://github.com/STEllAR-GROUP/hpx/pull/2095
\textsuperscript{2291} https://github.com/STEllAR-GROUP/hpx/pull/2094
\textsuperscript{2292} https://github.com/STEllAR-GROUP/hpx/issues/2093
\textsuperscript{2293} https://github.com/STEllAR-GROUP/hpx/pull/2091
\textsuperscript{2294} https://github.com/STEllAR-GROUP/hpx/pull/2090
\textsuperscript{2295} https://github.com/STEllAR-GROUP/hpx/pull/2089
\textsuperscript{2296} https://github.com/STEllAR-GROUP/hpx/issues/2088
\textsuperscript{2297} https://github.com/STEllAR-GROUP/hpx/pull/2087
\textsuperscript{2298} https://github.com/STEllAR-GROUP/hpx/issues/2086
\textsuperscript{2299} https://github.com/STEllAR-GROUP/hpx/pull/2085
\textsuperscript{2300} https://github.com/STEllAR-GROUP/hpx/pull/2084
\textsuperscript{2301} https://github.com/STEllAR-GROUP/hpx/pull/2083
\textsuperscript{2302} https://github.com/STEllAR-GROUP/hpx/pull/2082
\textsuperscript{2303} https://github.com/STEllAR-GROUP/hpx/pull/2081
\textsuperscript{2304} https://github.com/STEllAR-GROUP/hpx/pull/2080
\textsuperscript{2305} https://github.com/STEllAR-GROUP/hpx/issues/2079
\textsuperscript{2306} https://github.com/STEllAR-GROUP/hpx/issues/2078
\textsuperscript{2307} https://github.com/STEllAR-GROUP/hpx/pull/2077
\textsuperscript{2308} https://github.com/STEllAR-GROUP/hpx/pull/2076
• PR #2075<sup>2309</sup> - Fixed some documentation typos
• PR #2074<sup>2310</sup> - Adding flush-mode to message handler flush
• PR #2073<sup>2311</sup> - Fixing performance regression introduced lately
• PR #2072<sup>2312</sup> - Refactor local::condition_variable
• PR #2071<sup>2313</sup> - Timer based on boost::asio::deadline_timer
• PR #2070<sup>2314</sup> - Refactor tuple based functionality
• PR #2069<sup>2315</sup> - Fixed typos
• Issue #2068<sup>2316</sup> - Seg fault with octotiger
• PR #2067<sup>2317</sup> - Algorithm cleanup
• PR #2066<sup>2318</sup> - Split credit fixes
• PR #2065<sup>2319</sup> - Rename HPX_MOVABLE_BUT_NOT_COPYABLE to HPX_MOVABLE_ONLY
• PR #2064<sup>2320</sup> - Fixed some typos in docs
• PR #2063<sup>2321</sup> - Adding example demonstrating template components
• Issue #2062<sup>2322</sup> - Support component templates
• PR #2061<sup>2323</sup> - Replace some uses of lexical_cast<string> with C++11 std::to_string
• PR #2060<sup>2324</sup> - Replace uses of boost::noncopyable with HPX_NON_COPYABLE
• PR #2059<sup>2325</sup> - Adding missing for_loop algorithms
• PR #2058<sup>2326</sup> - Move several definitions to more appropriate headers
• PR #2057<sup>2327</sup> - Simplify assert_owns_lock and ignore_while_checking
• PR #2056<sup>2328</sup> - Replacing std::result_of with util::result_of
• PR #2055<sup>2329</sup> - Fix process launching/connecting back
• PR #2054<sup>2330</sup> - Add a forwarding coroutine header
• PR #2053<sup>2331</sup> - Replace uses of boost::unordered_map with std::unordered_map

<sup>2309</sup> https://github.com/STEllAR-GROUP/hpx/pull/2075
<sup>2310</sup> https://github.com/STEllAR-GROUP/hpx/pull/2074
<sup>2311</sup> https://github.com/STEllAR-GROUP/hpx/pull/2073
<sup>2312</sup> https://github.com/STEllAR-GROUP/hpx/pull/2072
<sup>2313</sup> https://github.com/STEllAR-GROUP/hpx/pull/2071
<sup>2314</sup> https://github.com/STEllAR-GROUP/hpx/pull/2070
<sup>2315</sup> https://github.com/STEllAR-GROUP/hpx/pull/2069
<sup>2316</sup> https://github.com/STEllAR-GROUP/hpx/issues/2068
<sup>2317</sup> https://github.com/STEllAR-GROUP/hpx/pull/2067
<sup>2318</sup> https://github.com/STEllAR-GROUP/hpx/pull/2066
<sup>2319</sup> https://github.com/STEllAR-GROUP/hpx/pull/2065
<sup>2320</sup> https://github.com/STEllAR-GROUP/hpx/pull/2064
<sup>2321</sup> https://github.com/STEllAR-GROUP/hpx/pull/2063
<sup>2322</sup> https://github.com/STEllAR-GROUP/hpx/issues/2062
<sup>2323</sup> https://github.com/STEllAR-GROUP/hpx/pull/2061
<sup>2324</sup> https://github.com/STEllAR-GROUP/hpx/pull/2060
<sup>2325</sup> https://github.com/STEllAR-GROUP/hpx/pull/2059
<sup>2326</sup> https://github.com/STEllAR-GROUP/hpx/pull/2058
<sup>2327</sup> https://github.com/STEllAR-GROUP/hpx/pull/2057
<sup>2328</sup> https://github.com/STEllAR-GROUP/hpx/pull/2056
<sup>2329</sup> https://github.com/STEllAR-GROUP/hpx/pull/2055
<sup>2330</sup> https://github.com/STEllAR-GROUP/hpx/pull/2054
<sup>2331</sup> https://github.com/STEllAR-GROUP/hpx/pull/2053
• PR #2052 - Rewrite tuple unwrap
• PR #2050 - Replace uses of BOOST_SCOPED_ENUM with C++11 scoped enums
• PR #2049 - Attempt to narrow down split_credit problem
• PR #2048 - Fixing gcc startup hangs
• PR #2047 - Fixing when_xxx and wait_xxx for MSVC12
• PR #2046 - adding persistent_auto_chunk_size and related tests for for_each
• PR #2045 - Fixing HPX_HAVE_THREAD_BACKTRACE_DEPTH build time configuration
• PR #2044 - Adding missing service executor types
• PR #2043 - Removing ambiguous definitions for is_future_range and future_range_traits
• PR #2042 - Clarify that HPX builds can use (much) more than 2GB per process
• PR #2041 - Changing future_iterator_traits to support pointers
• Issue #2040 - Improve documentation memory usage warning?
• PR #2039 - Coroutine cleanup
• PR #2038 - Fix cmake policy CMP0042 warning MACOSX_RPATH
• PR #2037 - Avoid redundant specialization of unique_function_nonser
• PR #2036 - nvcc dies with an internal error upon pushing/popping warnings inside templates
• Issue #2035 - Use a less restrictive iterator definition in hpx::cos::detail::future_iterator_traits
• PR #2034 - Fixing compilation error with thread queue wait time performance counter
• Issue #2033 - Compilation error when compiling with thread queue wait time performance counter
• Issue #2032 - Ambiguous template instantiation for is_future_range and future_range_traits.
• PR #2031 - Don’t restart timer on every incoming parcel
• PR #2030 - Unify handling of execution policies in parallel algorithms
• PR #2029 - Make pkg-config .pc files use .dylib on OSX

https://github.com/STEllAR-GROUP/hpx/pull/2052
https://github.com/STEllAR-GROUP/hpx/pull/2050
https://github.com/STEllAR-GROUP/hpx/pull/2049
https://github.com/STEllAR-GROUP/hpx/pull/2048
https://github.com/STEllAR-GROUP/hpx/pull/2047
https://github.com/STEllAR-GROUP/hpx/pull/2046
https://github.com/STEllAR-GROUP/hpx/pull/2045
https://github.com/STEllAR-GROUP/hpx/pull/2044
https://github.com/STEllAR-GROUP/hpx/pull/2043
https://github.com/STEllAR-GROUP/hpx/pull/2042
https://github.com/STEllAR-GROUP/hpx/pull/2041
https://github.com/STEllAR-GROUP/hpx/issues/2040
https://github.com/STEllAR-GROUP/hpx/pull/2039
https://github.com/STEllAR-GROUP/hpx/pull/2038
https://github.com/STEllAR-GROUP/hpx/pull/2037
https://github.com/STEllAR-GROUP/hpx/pull/2036
https://github.com/STEllAR-GROUP/hpx/issues/2035
https://github.com/STEllAR-GROUP/hpx/pull/2034
https://github.com/STEllAR-GROUP/hpx/issues/2033
https://github.com/STEllAR-GROUP/hpx/issues/2032
https://github.com/STEllAR-GROUP/hpx/pull/2031
https://github.com/STEllAR-GROUP/hpx/pull/2030
https://github.com/STEllAR-GROUP/hpx/pull/2029
- PR #2028 - Adding process component
- PR #2027 - Making check for compiler compatibility independent on compiler path
- PR #2025 - Fixing inspect tool
- PR #2024 - Intel13 removal
- PR #2023 - Fix errors related to older boost versions and parameter pack expansions in lambdas
- Issue #2022 - gmake fail: “No rule to make target /usr/lib46/libboost_context-mt.so”
- PR #2021 - Added Sudoku example
- Issue #2020 - Make errors related to init_globally.cpp example while building HPX out of the box
- PR #2019 - Fixed some compilation and cmake errors encountered in nqueen example
- PR #2018 - For loop algorithms
- PR #2017 - Non-recursive at_index implementation
- Issue #2016 - Add index-based for-loops
- Issue #2015 - Change default bind-mode to balanced
- PR #2014 - Fixed dataflow if invoked action returns a future
- PR #2013 - Fixing compilation issues with external example
- PR #2012 - Added Sierpinski Triangle example
- Issue #2011 - Compilation error while running sample hello_world_component code
- PR #2010 - Segmented move implemented for hpx::vector
- Issue #2009 - pkg-config order incorrect on 14.04 / GCC 4.8
- Issue #2008 - Compilation error in dataflow of action returning a future
- PR #2007 - Adding new performance counter exposing overall scheduler time
- PR #2006 - Function includes
- PR #2005 - Adding an example demonstrating how to initialize HPX from a global object

https://github.com/STEllAR-GROUP/hpx/pull/2028
https://github.com/STEllAR-GROUP/hpx/pull/2027
https://github.com/STEllAR-GROUP/hpx/pull/2025
https://github.com/STEllAR-GROUP/hpx/pull/2024
https://github.com/STEllAR-GROUP/hpx/pull/2023
https://github.com/STEllAR-GROUP/hpx/issues/2022
https://github.com/STEllAR-GROUP/hpx/pull/2021
https://github.com/STEllAR-GROUP/hpx/issues/2020
https://github.com/STEllAR-GROUP/hpx/pull/2019
https://github.com/STEllAR-GROUP/hpx/pull/2018
https://github.com/STEllAR-GROUP/hpx/pull/2017
https://github.com/STEllAR-GROUP/hpx/issues/2016
https://github.com/STEllAR-GROUP/hpx/issues/2015
https://github.com/STEllAR-GROUP/hpx/pull/2014
https://github.com/STEllAR-GROUP/hpx/pull/2013
https://github.com/STEllAR-GROUP/hpx/pull/2012
https://github.com/STEllAR-GROUP/hpx/issue/2011
https://github.com/STEllAR-GROUP/hpx/pull/2010
https://github.com/STEllAR-GROUP/hpx/issue/2008
https://github.com/STEllAR-GROUP/hpx/pull/2005
• PR #2004 - Fixing 2000
• PR #2003 - Adding generation parameter to gather to enable using it more than once
• PR #2003 - Turn on position independent code to solve link problem with hpx_init
• Issue #2001 - Gathering more than once segfaults
• Issue #2000 - Undefined reference to hpx::assertion_failed
• Issue #1999 - Seg fault in hpx::lcos::base_lco_with_value<*>::set_value_nonvirt() when running octo-tiger
• PR #1998 - Detect unknown command line options
• PR #1997 - Extending thread description
• PR #1996 - Adding natvis files to solution (MSVC only)
• Issue #1995 - Command line handling does not produce error
• PR #1994 - Possible missing include in test_utils.hpp
• PR #1993 - Add missing LANGUAGES tag to a hpx_add_compile_flag_if_available() call in CMakeLists.txt
• PR #1992 - Fixing shared_executor_test
• PR #1991 - Making sure the winsock library is properly initialized
• PR #1990 - Fixing bind_test placeholder ambiguity coming from boost-1.60
• PR #1989 - Performance tuning
• PR #1987 - Make configurable size of internal storage in util::function
• PR #1986 - AGAS Refactoring+1753 Cache mods
• PR #1985 - Adding missing task_block::run() overload taking an executor
• PR #1984 - Adding an optimized LRU Cache implementation (for AGAS)
• PR #1983 - Avoid invoking migration table look up for all objects
• PR #1981 - Replacing uintptr_t (which is not defined everywhere) with std::size_t
• PR #1980 - Optimizing LCO continuations

https://github.com/STEllAR-GROUP/hpx/pull/2004
https://github.com/STEllAR-GROUP/hpx/pull/2003
https://github.com/STEllAR-GROUP/hpx/pull/2002
https://github.com/STEllAR-GROUP/hpx/issues/2001
https://github.com/STEllAR-GROUP/hpx/issues/1999
https://github.com/STEllAR-GROUP/hpx/pull/1997
https://github.com/STEllAR-GROUP/hpx/pull/1996
https://github.com/STEllAR-GROUP/hpx/pull/1995
https://github.com/STEllAR-GROUP/hpx/pull/1994
https://github.com/STEllAR-GROUP/hpx/pull/1993
https://github.com/STEllAR-GROUP/hpx/pull/1992
https://github.com/STEllAR-GROUP/hpx/pull/1990
https://github.com/STEllAR-GROUP/hpx/pull/1989
https://github.com/STEllAR-GROUP/hpx/pull/1987
https://github.com/STEllAR-GROUP/hpx/pull/1986
https://github.com/STEllAR-GROUP/hpx/pull/1985
https://github.com/STEllAR-GROUP/hpx/pull/1984
https://github.com/STEllAR-GROUP/hpx/pull/1983
https://github.com/STEllAR-GROUP/hpx/pull/1981
• PR #1979 - Fixing Cori
• PR #1978 - Fix test check that got broken in hasty fix to memory overflow
• PR #1977 - Refactor action traits
• PR #1976 - Fixes typo in README.rst
• PR #1975 - Reduce size of benchmark timing arrays to fix test failures
• PR #1974 - Add action to update data owned by the partitioned_vector component
• PR #1973 - Adding partitioned_vector SPMD example
• PR #1971 - Fixing partitioned_vector SPMD example
• PR #1970 - Papi fixes
• PR #1969 - Fixing continuation recursions to not depend on fixed amount of recursions
• PR #1968 - More segmented algorithms
• Issue #1967 - Simplify component implementations
• PR #1966 - Migrate components
• Issue #1964 - fatal error: ‘boost(lockfree/detail/branch_hints.hpp’ file not found
• Issue #1962 - parallel:copy_if has race condition when used on in place arrays
• PR #1963 - Fixing Static Parcelport initialization
• PR #1961 - Fix function target
• Issue #1960 - Papi counters don’t reset
• PR #1959 - Fixing 1958
• Issue #1958 - inclusive_scan gives incorrect results with non-commutative operator
• PR #1957 - Fixing #1950
• PR #1956 - Sort by key example
• PR #1955 - Adding regression test for #1946: Hang in wait_all() in distributed run

2401 https://github.com/STEllAR-GROUP/hpx/pull/1979
2402 https://github.com/STEllAR-GROUP/hpx/pull/1978
2403 https://github.com/STEllAR-GROUP/hpx/pull/1977
2404 https://github.com/STEllAR-GROUP/hpx/pull/1976
2405 https://github.com/STEllAR-GROUP/hpx/pull/1975
2406 https://github.com/STEllAR-GROUP/hpx/pull/1974
2407 https://github.com/STEllAR-GROUP/hpx/pull/1972
2408 https://github.com/STEllAR-GROUP/hpx/pull/1971
2409 https://github.com/STEllAR-GROUP/hpx/pull/1970
2410 https://github.com/STEllAR-GROUP/hpx/pull/1969
2411 https://github.com/STEllAR-GROUP/hpx/pull/1968
2412 https://github.com/STEllAR-GROUP/hpx/issues/1967
2413 https://github.com/STEllAR-GROUP/hpx/pull/1966
2414 https://github.com/STEllAR-GROUP/hpx/issues/1964
2415 https://github.com/STEllAR-GROUP/hpx/issues/1962
2416 https://github.com/STEllAR-GROUP/hpx/pull/1963
2417 https://github.com/STEllAR-GROUP/hpx/pull/1961
2418 https://github.com/STEllAR-GROUP/hpx/issues/1960
2419 https://github.com/STEllAR-GROUP/hpx/pull/1959
2420 https://github.com/STEllAR-GROUP/hpx/issues/1958
2421 https://github.com/STEllAR-GROUP/hpx/pull/1957
2422 https://github.com/STEllAR-GROUP/hpx/pull/1956
2423 https://github.com/STEllAR-GROUP/hpx/pull/1955
• Issue #1954 - HPX releases should not use -Werror
• PR #1952 - Adding performance analysis for AGAS cache
• PR #1953 - Adapting test for explicit variadics to fail for gcc 4.6
• PR #1954 - Fixing memory leak
• Issue #1950 - Simplify external builds
• PR #1949 - Fixing yet another lock that is being held during suspension
• PR #1948 - Fixed container algorithms for Intel
• PR #1947 - Adding workaround for tagged_tuple
• Issue #1946 - Hang in wait_all() in distributed run
• PR #1945 - Fixed container algorithm tests
• Issue #1944 - assertion ‘p.destination_locality() == hpx::get_locality()’ failed
• PR #1943 - Fix a couple of compile errors with clang
• PR #1942 - Making parcel coalescing functional
• Issue #1941 - Re-enable parcel coalescing
• PR #1940 - Touching up make_future
• PR #1939 - Fixing problems in over-subscription management in the resource manager
• PR #1938 - Removing use of unified Boost.Thread header
• PR #1937 - Cleaning up the use of Boost.Accumulator headers
• PR #1936 - Making sure interval timer is started for aggregating performance counters
• PR #1935 - Tagged results
• PR #1934 - Fix remote async with deferred launch policy
• Issue #1933 - Floating point exception in statistics_counter<boost::accumulators::tag::mean>::get_counter_value
• PR #1932 - Removing superfluous includes of boost/lockfree/detail/branch_hints.hpp

2.11. Releases
• PR #1931\(^{2447}\) - fix compilation with clang 3.8.0
• Issue #1930\(^{2448}\) - Missing online documentation for HPX 0.9.11
• PR #1929\(^{2449}\) - LWG2485: get() should be overloaded for const tuple&&
• PR #1928\(^{2450}\) - Revert “Using ninja for circle-ci builds”
• PR #1927\(^{2451}\) - Using ninja for circle-ci builds
• PR #1926\(^{2452}\) - Fixing serialization of std::array
• Issue #1925\(^{2453}\) - Issues with static HPX libraries
• Issue #1924\(^{2454}\) - Performance degrading over time
• Issue #1923\(^{2455}\) - serialization of std::array appears broken in latest commit
• PR #1922\(^{2456}\) - Container algorithms
• PR #1921\(^{2457}\) - Tons of smaller quality improvements
• Issue #1920\(^{2458}\) - Seg fault in hpx::serialization::output_archive::add_gid when running octotiger
• Issue #1919\(^{2459}\) - Intel 15 compiler bug preventing HPX build
• PR #1918\(^{2460}\) - Address sanitizer fixes
• PR #1917\(^{2461}\) - Fixing compilation problems of parallel::sort with Intel compilers
• PR #1916\(^{2462}\) - Making sure code compiles if HPX_WITH_HWLOC=Off
• Issue #1915\(^{2463}\) - max_cores undefined if HPX_WITH_HWLOC=Off
• PR #1914\(^{2464}\) - Add utility member functions for partitioned_vector
• PR #1913\(^{2465}\) - Adding support for invoking actions to dataflow
• PR #1912\(^{2466}\) - Adding first batch of container algorithms
• PR #1910\(^{2467}\) - Keep cmake_module_path
• PR #1909\(^{2468}\) - Fix mpirun with pbs
• PR #1908\(^{2469}\) - Changing parallel::sort to return the last iterator as proposed by N4560

\(^{2447}\) https://github.com/STEllAR-GROUP/hpx/pull/1931
\(^{2448}\) https://github.com/STEllAR-GROUP/hpx/issues/1930
\(^{2449}\) https://github.com/STEllAR-GROUP/hpx/pull/1929
\(^{2450}\) https://github.com/STEllAR-GROUP/hpx/pull/1928
\(^{2451}\) https://github.com/STEllAR-GROUP/hpx/pull/1927
\(^{2452}\) https://github.com/STEllAR-GROUP/hpx/pull/1926
\(^{2453}\) https://github.com/STEllAR-GROUP/hpx/issues/1925
\(^{2454}\) https://github.com/STEllAR-GROUP/hpx/issues/1924
\(^{2455}\) https://github.com/STEllAR-GROUP/hpx/issues/1923
\(^{2456}\) https://github.com/STEllAR-GROUP/hpx/pull/1922
\(^{2457}\) https://github.com/STEllAR-GROUP/hpx/pull/1921
\(^{2458}\) https://github.com/STEllAR-GROUP/hpx/issues/1920
\(^{2459}\) https://github.com/STEllAR-GROUP/hpx/issues/1919
\(^{2460}\) https://github.com/STEllAR-GROUP/hpx/pull/1918
\(^{2461}\) https://github.com/STEllAR-GROUP/hpx/pull/1917
\(^{2462}\) https://github.com/STEllAR-GROUP/hpx/pull/1916
\(^{2463}\) https://github.com/STEllAR-GROUP/hpx/issues/1915
\(^{2464}\) https://github.com/STEllAR-GROUP/hpx/pull/1913
\(^{2465}\) https://github.com/STEllAR-GROUP/hpx/pull/1912
\(^{2466}\) https://github.com/STEllAR-GROUP/hpx/pull/1911
\(^{2467}\) https://github.com/STEllAR-GROUP/hpx/pull/1910
\(^{2468}\) https://github.com/STEllAR-GROUP/hpx/pull/1909
\(^{2469}\) https://github.com/STEllAR-GROUP/hpx/pull/1908
• PR #1907 - Adding a minimum version for Open MPI
• PR #1906 - Updates to the Release Procedure
• PR #1905 - Fixing #1903
• PR #1904 - Making sure std containers are cleared before serialization loads data
• Issue #1903 - When running octotiger, I get: assertion '(*new_gids_)[gid].size() == 1' failed: HPX(assertion_failure)
• Issue #1902 - Immediate crash when running hpx/octotiger with _GLIBCXX_DEBUG defined.
• PR #1901 - Making non-serializable classes non-serializable
• Issue #1900 - Two possible issues with std::list serialization
• PR #1899 - Fixing a problem with credit splitting as revealed by #1898
• Issue #1898 - Accessing component from locality where it was not created segfaults
• PR #1897 - Changing parallel::sort to return the last iterator as proposed by N4560
• Issue #1896 - version 1.0?
• Issue #1895 - Warning comment on numa_allocator is not very clear
• PR #1894 - Add support for compilers that have thread_local
• PR #1893 - Fixing 1890
• PR #1892 - Adds typed future_type for executor_traits
• PR #1891 - Fix wording in certain parallel algorithm docs
• Issue #1890 - Invoking papi counters give segfault
• PR #1889 - Fixing problems as reported by clang-check
• PR #1888 - WIP parallel is_heap
• PR #1887 - Fixed resetting performance counters related to idle-rate, etc
• Issue #1886 - Run hpx with qsub does not work
• PR #1885 - Warning cleaning pass

2.11. Releases
• PR #1884 Add missing parallel algorithm header
• PR #1883 Add feature test for thread_local on Clang for TLS
• PR #1882 Fix some redundant qualifiers
• Issue #1881 Unable to compile Octotiger using HPX and Intel MPI on SuperMIC
• Issue #1880 clang with libc++ on Linux needs TLS case
• PR #1879 Doc fixes for #1868
• PR #1878 Simplify functions
• PR #1877 Removing most usage of Boost.Config
• PR #1876 Add missing parallel algorithms to algorithm.hpp
• PR #1875 Simplify callables
• PR #1874 Address long standing FIXME on using std::unique_ptr with incomplete types
• PR #1873 Fixing 1871
• PR #1872 Making sure PBS environment uses specified node list even if no PBS_NODEFILE env is available
• Issue #1871 Fortran checks should be optional
• PR #1870 Touch local::mutex
• PR #1869 Documentation refactoring based off #1868
• PR #1867 Embrace static_assert
• PR #1866 Fix #1803 with documentation refactoring
• PR #1865 Setting OUTPUT_NAME as target properties
• PR #1863 Use SYSTEM for boost includes
• PR #1862 Minor cleanups
• PR #1861 Minor Corrections for Release
• PR #1860 Fixing hpx gdb script

https://github.com/STEllAR-GROUP/hpx/pull/1884
https://github.com/STEllAR-GROUP/hpx/pull/1883
https://github.com/STEllAR-GROUP/hpx/pull/1882
https://github.com/STEllAR-GROUP/hpx/issues/1881
https://github.com/STEllAR-GROUP/hpx/issues/1880
https://github.com/STEllAR-GROUP/hpx/pull/1879
https://github.com/STEllAR-GROUP/hpx/pull/1878
https://github.com/STEllAR-GROUP/hpx/pull/1877
https://github.com/STEllAR-GROUP/hpx/pull/1876
https://github.com/STEllAR-GROUP/hpx/pull/1875
https://github.com/STEllAR-GROUP/hpx/pull/1874
https://github.com/STEllAR-GROUP/hpx/pull/1873
https://github.com/STEllAR-GROUP/hpx/pull/1872
https://github.com/STEllAR-GROUP/hpx/issues/1871
https://github.com/STEllAR-GROUP/hpx/pull/1870
https://github.com/STEllAR-GROUP/hpx/pull/1869
https://github.com/STEllAR-GROUP/hpx/pull/1868
https://github.com/STEllAR-GROUP/hpx/pull/1867
https://github.com/STEllAR-GROUP/hpx/pull/1866
https://github.com/STEllAR-GROUP/hpx/pull/1865
https://github.com/STEllAR-GROUP/hpx/pull/1863
https://github.com/STEllAR-GROUP/hpx/pull/1862
https://github.com/STEllAR-GROUP/hpx/pull/1861
https://github.com/STEllAR-GROUP/hpx/pull/1860

Chapter 2. What’s so special about HPX?
• Issue #1859 - reset_active_counters resets times and thread counts before some of the counters are evaluated
• PR #1858 - Release V0.9.11
• PR #1857 - removing diskperf example from 9.11 release
• PR #1856 - fix return in packaged_task_base::reset()
• Issue #1842 - Install error: file INSTALL cannot find libhpx_parcel_coalescing.so.0.9.11
• PR #1839 - Adding fedora docs
• PR #1824 - Changing version on master to V0.9.12
• PR #1818 - Fixing #1748
• Issue #1815 - seg fault in AGAS
• Issue #1803 - wait_all documentation
• Issue #1796 - Outdated documentation to be revised
• Issue #1759 - glibc munmap_chunk or free(): invalid pointer on SuperMIC
• Issue #1753 - HPX performance degrades with time since execution begins
• Issue #1748 - All public HPX headers need to be self contained
• PR #1719 - How to build HPX with Visual Studio
• Issue #1684 - Race condition when using --hx:connect?
• PR #1658 - Add serialization for std::set (as there is for std::vector and std::map)
• PR #1641 - Generic client
• Issue #1632 - heartbeat example fails on separate nodes
• PR #1603 - Adds preferred namespace check to inspect tool
• Issue #1559 - Extend inspect tool
• Issue #1523 - Remote async with deferred launch policy never executes
• Issue #1472 - Serialization issues

2.11. Releases
• Issue #1457 - Implement N4392: C++ Latches and Barriers
• PR #1444 - Enabling usage of moveonly types for component construction
• Issue #1407 - The Intel 13 compiler has failing unit tests
• Issue #1405 - Allow component constructors to take movable only types
• Issue #1265 - Enable dataflow() to be usable with actions
• Issue #1236 - NUMA aware allocators
• Issue #802 - Fix Broken Examples
• Issue #559 - Add hpx::migrate facility
• Issue #449 - Make actions with template arguments usable and add documentation
• Issue #279 - Refactor addressing_service into a base class and two derived classes
• Issue #224 - Changing thread state metadata is not thread safe
• Issue #55 - Uniform syntax for enums should be implemented

2.11.9 HPX V0.9.11 (Nov 11, 2015)

Our main focus for this release was the design and development of a coherent set of higher-level APIs exposing various types of parallelism to the application programmer. We introduced the concepts of an executor, which can be used to customize the where and when of execution of tasks in the context of parallelizing codes. We extended all APIs related to managing parallel tasks to support executors which gives the user the choice of either using one of the predefined executor types or to provide its own, possibly application specific, executor. We paid very close attention to align all of these changes with the existing C++ Standards documents or with the ongoing proposals for standardization.

This release is the first after our change to a new development policy. We switched all development to be strictly performed on branches only, all direct commits to our main branch (master) are prohibited. Any change has to go through a peer review before it will be merged to master. As a result the overall stability of our code base has significantly increased, the development process itself has been simplified. This change manifests itself in a large number of pull-requests which have been merged (please see below for a full list of closed issues and pull-requests). All in all for this release, we closed almost 100 issues and merged over 290 pull-requests. There have been over 1600 commits to the master branch since the last release.

General changes

• We are moving into the direction of unifying managed and simple components. As such, the classes hpx::components::component and hpx::components::component_base have been added which currently just forward to the currently existing simple component facilities. The examples have been converted to only use those two classes.

2539 https://github.com/STEllAR-GROUP/hpx/issues/1457
2540 https://github.com/STEllAR-GROUP/hpx/pull/1444
2541 https://github.com/STEllAR-GROUP/hpx/issues/1407
2542 https://github.com/STEllAR-GROUP/hpx/issues/1405
2543 https://github.com/STEllAR-GROUP/hpx/issues/1265
2544 https://github.com/STEllAR-GROUP/hpx/issues/1236
2545 https://github.com/STEllAR-GROUP/hpx/issues/802
2546 https://github.com/STEllAR-GROUP/hpx/issues/559
2547 https://github.com/STEllAR-GROUP/hpx/issues/449
2548 https://github.com/STEllAR-GROUP/hpx/issues/279
2549 https://github.com/STEllAR-GROUP/hpx/issues/224
2550 https://github.com/STEllAR-GROUP/hpx/issues/55
• Added integration with the CircleCI\textsuperscript{2551} hosted continuous integration service. This gives us constant and immediate feedback on the health of our master branch.

• The compiler configuration subsystem in the build system has been reimplemented. Instead of using Boost.Config we now use our own lightweight set of cmake scripts to determine the available language and library features supported by the used compiler.

• The API for creating instances of components has been consolidated. All component instances should be created using the `hpx::new_` only. It allows to instantiate both, single component instances and multiple component instances. The placement of the created components can be controlled by special distribution policies. Please see the corresponding documentation outlining the use of `hpx::new_`.

• Introduced four new distribution policies which can be used with many API functions which traditionally expected to be used with a locality id. The new distribution policies are:
  
  \begin{itemize}
    \item `hpx::components::default_distribution_policy` which tries to place multiple component instances as evenly as possible.
    \item `hpx::components::colocating_distribution_policy` which will refer to the locality where a given component instance is currently placed.
    \item `hpx::components::binpacking_distribution_policy` which will place multiple component instances as evenly as possible based on any performance counter.
    \item `hpx::components::target_distribution_policy` which allows to represent a given locality in the context of a distribution policy.
  \end{itemize}

• The new distribution policies can now be also used with `hpx::async`. This change also deprecates `hpx::async_colocated(id, ...)` which now is replaced by a distribution policy: `hpx::async(hpx::colocated(id), ...)`.  

• The `hpx::vector` and `hpx::unordered_map` data structures can now be used with the new distribution policies as well.

• The parallel facility `hpx::parallel::task_region` has been renamed to `hpx::parallel::task_block` based on the changes in the corresponding standardization proposal N4411\textsuperscript{2552}.

• Added extensions to the parallel facility `hpx::parallel::task_block` allowing to combine a `task_block` with an execution policy. This implies a minor breaking change as the `hpx::parallel::task_block` is now a template.

• Added new LCOs: `hpx::lcos::latch` and `hpx::lcos::local::latch` which semantically conform to the proposed `std::latch` (see N4399\textsuperscript{2553}).

• Added performance counters exposing data related to data transferred by input/output (filesystem) operations (thanks to Maciej Brodowicz).

• Added performance counters allowing to track the number of action invocations (local and remote invocations).

• Added new command line options `--hpx:print-counter-at` and `--hpx:reset-counters`.

• The `hpx::vector` component has been renamed to `hpx::partitioned_vector` to make it explicit that the underlying memory is not contiguous.

• Introduced a completely new and uniform higher-level parallelism API which is based on executors. All existing parallelism APIs have been adapted to this. We have added a large number of different executor types, such as a numa-aware executor, a this-thread executor, etc.

• Added support for the MingW toolchain on Windows (thanks to Eric Lemanissier).

\textsuperscript{2551} https://circleci.com/gh/STEllAR-GROUP/hpx
\textsuperscript{2552} http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2015/n4411.pdf
\textsuperscript{2553} http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2015/n4399.html
HPX Documentation, 1.4.1

- HPX now includes support for APEX, (Autonomic Performance Environment for eXascale). APEX is an instrumentation and software adaptation library that provides an interface to TAU profiling / tracing as well as runtime adaptation of HPX applications through policy definitions. For more information and documentation, please see https://github.com/khuck/xpress-apex. To enable APEX at configuration time, specify -DHPX_WITH_APEX=On. To also include support for TAU profiling, specify -DHPX_WITH_TAU=On and specify the -DTAU_ROOT, -DTAU_ARCH and -DTAU_OPTIONS cmake parameters.

- We have implemented many more of the Using parallel algorithms. Please see Issue #1141 for the list of all available parallel algorithms (thanks to Daniel Bourgeois and John Biddiscombe for contributing their work).

Breaking changes

- We are moving into the direction of unifying managed and simple components. In order to stop exposing the old facilities, all examples have been converted to use the new classes. The breaking change in this release is that performance counters are now a hpx::components::component_base instead of hpx::components::managed_component_base.

- We removed the support for stackless threads. It turned out that there was no performance benefit when using stackless threads. As such, we decided to clean up our codebase. This feature was not documented.

- The CMake project name has changed from `hpx` to ‘HPX’ for consistency and compatibility with naming conventions and other CMake projects. Generated config files go into <prefix>/lib/cmake/HPX and not <prefix>/lib/cmake/hpx.

- The macro HPX_REGISTER_MINIMAL_COMPONENT_FACTORY has been deprecated. Please use HPX_REGISTER_COMPONENT, instead. The old macro will be removed in the next release.

- The obsolete distributing_factory and binpacking_factory components have been removed. The corresponding functionality is now provided by the hpx::new_ API function in conjunction with the hpx::default_layout and hpx::binpacking distribution policies (hpx::components::default_distribution_policy and hpx::components::binpacking_distribution_policy)

- The API function hpx::new_colocated has been deprecated. Please use the consolidated API hpx::new in conjunction with the new hpx::colocated distribution policy (hpx::components::colocating_distribution_policy) instead. The old API function will still be available for at least one release of HPX if the configuration variable HPX_WITH_COLOCATED_BACKWARDS_COMPATIBILITY is enabled.

- The API function hpx::async_colocated has been deprecated. Please use the consolidated API hpx::async in conjunction with the new hpx::colocated distribution policy (hpx::components::colocating_distribution_policy) instead. The old API function will still be available for at least one release of HPX if the configuration variable HPX_WITH_COLOCATED_BACKWARDS_COMPATIBILITY is enabled.

- The obsolete remote_object component has been removed.

- Replaced the use of Boost.Serialization with our own solution. While the new version is mostly compatible with Boost.Serialization, this change requires some minor code modifications in user code. For more information, please see the corresponding announcement on the hpx-users@stellar.cct.lsu.edu mailing list.

- The names used by cmake to influence various configuration options have been unified. The new naming scheme relies on all configuration constants to start with HPX_WITH..., while the preprocessor constant which is used at build time starts with HPX_HAVE.... For instance, the former cmake command line -DHPX_MALLOC=... now has to be specified a -DHPX_HAVE_MALLOC=... and will cause the preprocessor constant HPX_HAVE_MALLOC to be defined. The actual name of the constant (i.e. MALLOC) has not

---

2554 https://github.com/STEllAR-GROUP/hpx/issues/1141
2555 http://thread.gmane.org/gmane.comp.lib.hpx.devel/196
changed. Please see the corresponding documentation for more details (*CMake variables used to configure HPX*).

- The `get_gid()` functions exposed by the component base classes `hpx::components::server::simple_component_base`, `hpx::components::server::managed_component_base`, and `hpx::components::server::fixed_component_base` have been replaced by two new functions: `get_unmanaged_id()` and `get_id()`. To enable the old function name for backwards compatibility, use the cmake configuration option `HPX_WITH_COMPONENT_GET_GID_COMPATIBILITY=On`.

- All functions which were named `get_gid()` but were returning `hpx::id_type` have been renamed to `get_id()`. To enable the old function names for backwards compatibility, use the cmake configuration option `HPX_WITH_COMPONENT_GET_GID_COMPATIBILITY=On`.

**Bug fixes (closed tickets)**

Here is a list of the important tickets we closed for this release.

- PR #1855\(^{2556}\) - Completely removing external/endian
- PR #1854\(^{2557}\) - Don’t pollute CMAKE_CXX_FLAGS through find_package()
- PR #1853\(^{2558}\) - Updating CMakе configuration to get correct version of TAU library
- PR #1852\(^{2559}\) - Fixing Performance Problems with MPI Parcelport
- PR #1851\(^{2560}\) - Fixing hpx_add_link_flag() and hpx_remove_link_flag()
- PR #1850\(^{2561}\) - Fixing 1836, adding parallel::sort
- PR #1849\(^{2562}\) - Fixing configuration for use of more than 64 cores
- PR #1848\(^{2563}\) - Change default APEX version for release
- PR #1847\(^{2564}\) - Fix client_base::then on release
- PR #1846\(^{2565}\) - Removing broken lcos::local::channel from release
- PR #1845\(^{2566}\) - Adding example demonstrating a possible safe-object implementation to release
- PR #1844\(^{2567}\) - Removing stubs from accumulator examples
- PR #1843\(^{2568}\) - Don’t pollute CMAKE_CXX_FLAGS through find_package()
- PR #1841\(^{2569}\) - Fixing client_base<>::then
- PR #1840\(^{2570}\) - Adding example demonstrating a possible safe-object implementation
- PR #1838\(^{2571}\) - Update version rc1

\(^{2556}\) https://github.com/STEllAR-GROUP/hpx/pull/1855
\(^{2557}\) https://github.com/STEllAR-GROUP/hpx/pull/1854
\(^{2558}\) https://github.com/STEllAR-GROUP/hpx/pull/1853
\(^{2559}\) https://github.com/STEllAR-GROUP/hpx/pull/1852
\(^{2560}\) https://github.com/STEllAR-GROUP/hpx/pull/1851
\(^{2561}\) https://github.com/STEllAR-GROUP/hpx/pull/1850
\(^{2562}\) https://github.com/STEllAR-GROUP/hpx/pull/1849
\(^{2563}\) https://github.com/STEllAR-GROUP/hpx/pull/1848
\(^{2564}\) https://github.com/STEllAR-GROUP/hpx/pull/1847
\(^{2565}\) https://github.com/STEllAR-GROUP/hpx/pull/1846
\(^{2566}\) https://github.com/STEllAR-GROUP/hpx/pull/1845
\(^{2567}\) https://github.com/STEllAR-GROUP/hpx/pull/1844
\(^{2568}\) https://github.com/STEllAR-GROUP/hpx/pull/1843
\(^{2569}\) https://github.com/STEllAR-GROUP/hpx/pull/1841
\(^{2570}\) https://github.com/STEllAR-GROUP/hpx/pull/1840
\(^{2571}\) https://github.com/STEllAR-GROUP/hpx/pull/1838
• PR #1837\textsuperscript{2572} - Removing broken lcos::local::channel
• PR #1835\textsuperscript{2573} - Adding explicit move constructor and assignment operator to hpx::lcos::promise
• PR #1834\textsuperscript{2574} - Making hpx::lcos::promise move-only
• PR #1833\textsuperscript{2575} - Adding fedora docs
• Issue #1832\textsuperscript{2576} - hpx::lcos::promise<> must be move-only
• PR #1831\textsuperscript{2577} - Fixing resource manager gcc5.2
• PR #1830\textsuperscript{2578} - Fix intel13
• PR #1829\textsuperscript{2579} - Unbreaking thread test
• PR #1828\textsuperscript{2580} - Fixing #1620
• PR #1827\textsuperscript{2581} - Fixing a memory management issue for the Parquet application
• Issue #1826\textsuperscript{2582} - Memory management issue in hpx::lcos::promise
• PR #1825\textsuperscript{2583} - Adding hpx::components::component and hpx::components::component_base
• PR #1823\textsuperscript{2584} - Adding git commit id to circeci build
• PR #1822\textsuperscript{2585} - applying fixes suggested by clang 3.7
• PR #1821\textsuperscript{2586} - Hyperlink fixes
• PR #1820\textsuperscript{2587} - added parallel multi-locality sanity test
• PR #1819\textsuperscript{2588} - Fixing #1667
• Issue #1817\textsuperscript{2589} - Hyperlinks generated by inspect tool are wrong
• PR #1816\textsuperscript{2590} - Support hpxrx
• PR #1814\textsuperscript{2591} - Fix async to dispatch to the correct locality in all cases
• Issue #1813\textsuperscript{2592} - async(launch::... , action(), ...) always invokes locally
• PR #1812\textsuperscript{2593} - fixed syntax error in CMakeLists.txt
• PR #1811\textsuperscript{2594} - Agas optimizations

\textsuperscript{2572} https://github.com/STEllAR-GROUP/hpx/pull/1837
\textsuperscript{2573} https://github.com/STEllAR-GROUP/hpx/pull/1835
\textsuperscript{2574} https://github.com/STEllAR-GROUP/hpx/pull/1834
\textsuperscript{2575} https://github.com/STEllAR-GROUP/hpx/pull/1833
\textsuperscript{2576} https://github.com/STEllAR-GROUP/hpx/issues/1832
\textsuperscript{2577} https://github.com/STEllAR-GROUP/hpx/pull/1831
\textsuperscript{2578} https://github.com/STEllAR-GROUP/hpx/pull/1830
\textsuperscript{2579} https://github.com/STEllAR-GROUP/hpx/pull/1829
\textsuperscript{2580} https://github.com/STEllAR-GROUP/hpx/pull/1828
\textsuperscript{2581} https://github.com/STEllAR-GROUP/hpx/pull/1827
\textsuperscript{2582} https://github.com/STEllAR-GROUP/hpx/issues/1826
\textsuperscript{2583} https://github.com/STEllAR-GROUP/hpx/pull/1825
\textsuperscript{2584} https://github.com/STEllAR-GROUP/hpx/pull/1823
\textsuperscript{2585} https://github.com/STEllAR-GROUP/hpx/pull/1822
\textsuperscript{2586} https://github.com/STEllAR-GROUP/hpx/pull/1821
\textsuperscript{2587} https://github.com/STEllAR-GROUP/hpx/pull/1820
\textsuperscript{2588} https://github.com/STEllAR-GROUP/hpx/pull/1819
\textsuperscript{2589} https://github.com/STEllAR-GROUP/hpx/issues/1817
\textsuperscript{2590} https://github.com/STEllAR-GROUP/hpx/pull/1816
\textsuperscript{2591} https://github.com/STEllAR-GROUP/hpx/pull/1814
\textsuperscript{2592} https://github.com/STEllAR-GROUP/hpx/issues/1813
\textsuperscript{2593} https://github.com/STEllAR-GROUP/hpx/pull/1812
\textsuperscript{2594} https://github.com/STEllAR-GROUP/hpx/pull/1811
• PR #1810 - drop superfluous typedefs
• PR #1809 - Allow HPX to be used as an optional package in 3rd party code
• PR #1808 - Fixing #1723
• PR #1807 - Making sure resolve_localities does not hang during normal operation
• Issue #1806 - Spinlock no longer movable and deletes operator ‘=’, breaks MiniGhost
• Issue #1804 - register_with_basename causes hangs
• PR #1801 - Enhanced the inspect tool to take user directly to the problem with hyperlinks
• Issue #1800 - Problems compiling application on smic
• PR #1799 - Fixing cv exceptions
• PR #1798 - Documentation refactoring & updating
• PR #1797 - Updating the activeharmony CMake module
• PR #1795 - Fixing cv
• PR #1794 - Fix connect with hpx::runtime_mode_connect
• PR #1793 - fix a wrong use of HPX_MAX_CPU_COUNT instead of HPX_HA VE_MAX_CPU_COUNT
• PR #1792 - Allow for default constructed parcel instances to be moved
• PR #1791 - Fix connect with hpx::runtime_mode_connect
• Issue #1790 - assertion action_.get() failed: HPX(assertion_failure) when running Octotiger with pull request 1786
• PR #1789 - Fixing discover_counter_types API function
• Issue #1788 - connect with hpx::runtime_mode_connect
• Issue #1787 - discover_counter_types not working
• PR #1786 - Changing addressing_service to use std::unordered_map instead of std::map
• PR #1785 - Fix is_iterator for container algorithms
• PR #1784 - Adding new command line options:

---

2.11. Releases

---

2595 https://github.com/STEllAR-GROUP/hpx/pull/1810
2596 https://github.com/STEllAR-GROUP/hpx/pull/1809
2597 https://github.com/STEllAR-GROUP/hpx/pull/1808
2598 https://github.com/STEllAR-GROUP/hpx/pull/1807
2599 https://github.com/STEllAR-GROUP/hpx/issues/1806
2600 https://github.com/STEllAR-GROUP/hpx/issues/1804
2601 https://github.com/STEllAR-GROUP/hpx/pull/1801
2602 https://github.com/STEllAR-GROUP/hpx/issues/1800
2603 https://github.com/STEllAR-GROUP/hpx/pull/1799
2604 https://github.com/STEllAR-GROUP/hpx/pull/1798
2605 https://github.com/STEllAR-GROUP/hpx/pull/1797
2606 https://github.com/STEllAR-GROUP/hpx/pull/1795
2607 https://github.com/STEllAR-GROUP/hpx/pull/1794
2608 https://github.com/STEllAR-GROUP/hpx/pull/1793
2609 https://github.com/STEllAR-GROUP/hpx/pull/1792
2610 https://github.com/STEllAR-GROUP/hpx/pull/1791
2611 https://github.com/STEllAR-GROUP/hpx/issues/1790
2612 https://github.com/STEllAR-GROUP/hpx/pull/1789
2613 https://github.com/STEllAR-GROUP/hpx/issues/1788
2614 https://github.com/STEllAR-GROUP/hpx/issues/1787
2615 https://github.com/STEllAR-GROUP/hpx/pull/1786
2616 https://github.com/STEllAR-GROUP/hpx/pull/1785
2617 https://github.com/STEllAR-GROUP/hpx/pull/1784
• PR #1783 - Minor changes for APEX support
• PR #1782 - Drop legacy forwarding action traits
• PR #1781 - Attempt to resolve the race between cv::wait_xxx and cv::notify_all
• PR #1780 - Removing serialize_sequence
• PR #1779 - Fixed #1501: hwloc configuration options are wrong for MIC
• PR #1778 - Removing ability to enable/disable parcel handling
• PR #1777 - Completely removing stackless threads
• PR #1776 - Cleaning up util/plugin
• PR #1775 - Agas fixes
• PR #1774 - Action invocation count
• PR #1773 - replaced MSVC variable with WIN32
• PR #1772 - Fixing Problems in MPI parcelport and future serialization.
• PR #1771 - Fixing intel 13 compiler errors related to variadic template template parameters for lcos::when_tests
• PR #1770 - Forwarding decay to std::
• PR #1769 - Add more characters with special regex meaning to the existing patch
• PR #1768 - Adding test for receive_buffer
• PR #1767 - Making sure that uptime counter throws exception on any attempt to be reset
• PR #1766 - Cleaning up code related to throttling scheduler
• PR #1765 - Restricting thread_data to creating only with intrusive_pointers
• PR #1764 - Fixing 1763
• Issue #1763 - UB in thread_data::operator delete
• PR #1762 - Making sure all serialization registries/factories are unique
• PR #1761 - Fixed #1751: hpx::future::wait_for fails a simple test

https://github.com/STEllAR-GROUP/hpx/pull/1783
https://github.com/STEllAR-GROUP/hpx/pull/1782
https://github.com/STEllAR-GROUP/hpx/pull/1781
https://github.com/STEllAR-GROUP/hpx/pull/1780
https://github.com/STEllAR-GROUP/hpx/pull/1779
https://github.com/STEllAR-GROUP/hpx/pull/1778
https://github.com/STEllAR-GROUP/hpx/pull/1777
https://github.com/STEllAR-GROUP/hpx/pull/1776
https://github.com/STEllAR-GROUP/hpx/pull/1775
https://github.com/STEllAR-GROUP/hpx/pull/1774
https://github.com/STEllAR-GROUP/hpx/pull/1773
https://github.com/STEllAR-GROUP/hpx/pull/1772
https://github.com/STEllAR-GROUP/hpx/pull/1771
https://github.com/STEllAR-GROUP/hpx/pull/1770
https://github.com/STEllAR-GROUP/hpx/pull/1769
https://github.com/STEllAR-GROUP/hpx/pull/1768
https://github.com/STEllAR-GROUP/hpx/pull/1767
https://github.com/STEllAR-GROUP/hpx/pull/1766
https://github.com/STEllAR-GROUP/hpx/pull/1765
https://github.com/STEllAR-GROUP/hpx/pull/1764
https://github.com/STEllAR-GROUP/hpx/issues/1763
https://github.com/STEllAR-GROUP/hpx/pull/1762
https://github.com/STEllAR-GROUP/hpx/pull/1761
• PR #1758 - Fixing #1757
• Issue #1757 - pinning not correct using -hpx:bind
• Issue #1756 - compilation error with MinGW
• PR #1755 - Making output serialization const-correct
• Issue #1753 - HPX performance degrades with time since execution begins
• Issue #1752 - Error in AGAS
• Issue #1751 - hpx::future::wait_for fails a simple test
• PR #1750 - Removing hpx_fwd.hpp includes
• PR #1749 - Simplify result_of and friends
• PR #1747 - Removed superfluous code from message_buffer.hpp
• PR #1746 - Tuple dependencies
• Issue #1745 - Broken when_some which takes iterators
• PR #1744 - Refining archive interface
• PR #1743 - Fixing when_all when only a single future is passed
• PR #1742 - Config includes
• PR #1741 - Os executors
• Issue #1740 - hpx::promise has some problems
• PR #1739 - Parallel composition with generic containers
• Issue #1738 - After building program and successfully linking to a version of hpx DHPX_DIR seems to be ignored
• Issue #1737 - Uptime problems
• PR #1736 - added convenience c-tor and begin()/end() to serialize_buffer
• PR #1735 - Config includes
• PR #1734 - Fixed #1688: Add timer counters for tfunc_total and exec_total

https://github.com/STEllAR-GROUP/hpx/pull/1758
https://github.com/STEllAR-GROUP/hpx/commits/1757
https://github.com/STEllAR-GROUP/hpx/commits/1756
https://github.com/STEllAR-GROUP/hpx/commits/1755
https://github.com/STEllAR-GROUP/hpx/commits/1753
https://github.com/STEllAR-GROUP/hpx/commits/1752
https://github.com/STEllAR-GROUP/hpx/commits/1751
https://github.com/STEllAR-GROUP/hpx/commits/1750
https://github.com/STEllAR-GROUP/hpx/commits/1749
https://github.com/STEllAR-GROUP/hpx/commits/1748
https://github.com/STEllAR-GROUP/hpx/commits/1747
https://github.com/STEllAR-GROUP/hpx/commits/1746
https://github.com/STEllAR-GROUP/hpx/commits/1745
https://github.com/STEllAR-GROUP/hpx/commits/1744
https://github.com/STEllAR-GROUP/hpx/commits/1743
https://github.com/STEllAR-GROUP/hpx/commits/1742
https://github.com/STEllAR-GROUP/hpx/commits/1741
https://github.com/STEllAR-GROUP/hpx/commits/1740
https://github.com/STEllAR-GROUP/hpx/commits/1739
https://github.com/STEllAR-GROUP/hpx/commits/1738
https://github.com/STEllAR-GROUP/hpx/commits/1737
https://github.com/STEllAR-GROUP/hpx/commits/1736
https://github.com/STEllAR-GROUP/hpx/commits/1735
https://github.com/STEllAR-GROUP/hpx/commits/1734

2.11. Releases
• Issue #1733 - Add unit test for hpx/lcos/local/receive_buffer.hpp
• PR #1732 - Renaming get_os_thread_count
• PR #1731 - Basename registration
• Issue #1730 - Use after move of thread_init_data
• PR #1729 - Fixing #1722
• PR #1727 - Fixing compile problems with apply_colocated
• PR #1726 - Apex integration
• PR #1725 - fixed test timeouts
• PR #1724 - Renaming vector
• Issue #1723 - Drop support for intel compilers and gcc 4.4. based standard libs
• Issue #1722 - Add support for detecting non-ready futures before serialization
• PR #1721 - Unifying parallel executors, initializing from launch policy
• PR #1720 - dropped superfluous typedef
• Issue #1718 - Windows 10 x64, VS 2015 - Unknown CMake command “add_hpx_pseudo_target”.
• PR #1717 - Timed executor traits for thread-executors
• PR #1716 - serialization of arrays didn’t work with non-pod types. fixed
• PR #1715 - List serialization
• PR #1714 - changing misspellings
• PR #1713 - Fixed distribution policy executors
• PR #1712 - Moving library detection to be executed after feature tests
• PR #1711 - Simplify parcel
• PR #1710 - Compile only tests

https://github.com/STEllAR-GROUP/hpx/issues/1733
https://github.com/STEllAR-GROUP/hpx/pull/1732
https://github.com/STEllAR-GROUP/hpx/pull/1731
https://github.com/STEllAR-GROUP/hpx/issues/1730
https://github.com/STEllAR-GROUP/hpx/pull/1729
https://github.com/STEllAR-GROUP/hpx/pull/1728
https://github.com/STEllAR-GROUP/hpx/pull/1727
https://github.com/STEllAR-GROUP/hpx/pull/1726
https://github.com/STEllAR-GROUP/hpx/pull/1725
https://github.com/STEllAR-GROUP/hpx/pull/1724
https://github.com/STEllAR-GROUP/hpx/issues/1723
https://github.com/STEllAR-GROUP/hpx/issues/1722
https://github.com/STEllAR-GROUP/hpx/pull/1721
https://github.com/STEllAR-GROUP/hpx/pull/1720
https://github.com/STEllAR-GROUP/hpx/pull/1718
https://github.com/STEllAR-GROUP/hpx/pull/1717
https://github.com/STEllAR-GROUP/hpx/pull/1716
https://github.com/STEllAR-GROUP/hpx/pull/1715
https://github.com/STEllAR-GROUP/hpx/pull/1714
https://github.com/STEllAR-GROUP/hpx/pull/1713
https://github.com/STEllAR-GROUP/hpx/pull/1712
https://github.com/STEllAR-GROUP/hpx/pull/1711
https://github.com/STEllAR-GROUP/hpx/pull/1710
• PR #1709\(^{2687}\) - Implemented timed executors
• PR #1708\(^{2688}\) - Implement parallel::executor_traits for thread-executors
• PR #1707\(^{2689}\) - Various fixes to threads::executors to make custom schedulers work
• PR #1706\(^{2690}\) - Command line option –hpx:cores does not work as expected
• Issue #1705\(^{2691}\) - command line option –hpx:cores does not work as expected
• PR #1704\(^{2692}\) - vector deserialization is speeded up a little
• PR #1703\(^{2693}\) - Fixing shared_mutes
• Issue #1702\(^{2694}\) - Shared_mutex does not compile with no_mutex cond_var
• PR #1701\(^{2695}\) - Add distribution_policy_executor
• PR #1700\(^{2696}\) - Executor parameters
• PR #1699\(^{2697}\) - Readers writer lock
• PR #1698\(^{2698}\) - Remove leftovers
• PR #1697\(^{2699}\) - Fixing held locks
• PR #1696\(^{2700}\) - Modified Scan Partitioner for Algorithms
• PR #1695\(^{2701}\) - This thread executors
• PR #1694\(^{2702}\) - Fixed #1688: Add timer counters for tfunc_total and exec_total
• PR #1693\(^{2703}\) - Fix #1691: is_executor template specification fails for inherited executors
• PR #1692\(^{2704}\) - Fixed #1662: Possible exception source in coalescing_message_handler
• Issue #1691\(^{2705}\) - is_executor template specification fails for inherited executors
• PR #1690\(^{2706}\) - added macro for non-intrusive serialization of classes without a default c-tor
• PR #1689\(^{2707}\) - Replace value_or_error with custom storage, unify future_data state
• Issue #1688\(^{2708}\) - Add timer counters for tfunc_total and exec_total
• PR #1687\(^{2709}\) - Fixed interval timer

\(^{2687}\) https://github.com/STEllAR-GROUP/hpx/pull/1709
\(^{2688}\) https://github.com/STEllAR-GROUP/hpx/pull/1708
\(^{2689}\) https://github.com/STEllAR-GROUP/hpx/pull/1707
\(^{2690}\) https://github.com/STEllAR-GROUP/hpx/pull/1706
\(^{2691}\) https://github.com/STEllAR-GROUP/hpx/issues/1705
\(^{2692}\) https://github.com/STEllAR-GROUP/hpx/pull/1704
\(^{2693}\) https://github.com/STEllAR-GROUP/hpx/pull/1703
\(^{2694}\) https://github.com/STEllAR-GROUP/hpx/issues/1702
\(^{2695}\) https://github.com/STEllAR-GROUP/hpx/pull/1701
\(^{2696}\) https://github.com/STEllAR-GROUP/hpx/pull/1700
\(^{2697}\) https://github.com/STEllAR-GROUP/hpx/pull/1699
\(^{2698}\) https://github.com/STEllAR-GROUP/hpx/pull/1698
\(^{2699}\) https://github.com/STEllAR-GROUP/hpx/pull/1697
\(^{2700}\) https://github.com/STEllAR-GROUP/hpx/pull/1696
\(^{2701}\) https://github.com/STEllAR-GROUP/hpx/pull/1695
\(^{2702}\) https://github.com/STEllAR-GROUP/hpx/pull/1694
\(^{2703}\) https://github.com/STEllAR-GROUP/hpx/pull/1693
\(^{2704}\) https://github.com/STEllAR-GROUP/hpx/pull/1692
\(^{2705}\) https://github.com/STEllAR-GROUP/hpx/issues/1691
\(^{2706}\) https://github.com/STEllAR-GROUP/hpx/pull/1690
\(^{2707}\) https://github.com/STEllAR-GROUP/hpx/pull/1689
\(^{2708}\) https://github.com/STEllAR-GROUP/hpx/issues/1688
\(^{2709}\) https://github.com/STEllAR-GROUP/hpx/pull/1687

2.11. Releases
- PR #1686 - Fixing cmake warnings about not existing pseudo target dependencies
- PR #1685 - Converting partitioners to use bulk async execute
- PR #1683 - Adds a tool for inspect that checks for character limits
- PR #1682 - Change project name to (uppercase) HPX
- PR #1681 - Counter shortnames
- PR #1680 - Extended Non-intrusive Serialization to Ease Usage for Library Developers
- PR #1679 - Working on 1544: More executor changes
- PR #1678 - Transpose fixes
- PR #1677 - Improve Boost compatibility check
- PR #1676 - 1d stencil fix
- Issue #1675 - hpx project name is not HPX
- PR #1674 - Fixing the MPI parcelport
- PR #1673 - added move semantics to map/vector deserialization
- PR #1672 - Vs2015 await
- PR #1671 - Adapt transform for #1668
- PR #1670 - Started to work on #1668
- PR #1669 - Add this_thread_executors
- Issue #1667 - Apple build instructions in docs are out of date
- PR #1666 - Apex integration
- PR #1665 - Fixes an error with the whitespace check that showed the incorrect location of the error
- Issue #1664 - Inspect tool found incorrect endline whitespace
- PR #1663 - Improve use of locks
- Issue #1662 - Possible exception source in coalescing_message_handler

2710 https://github.com/STEllAR-GROUP/hpx/pull/1686
2711 https://github.com/STEllAR-GROUP/hpx/pull/1685
2712 https://github.com/STEllAR-GROUP/hpx/pull/1683
2713 https://github.com/STEllAR-GROUP/hpx/pull/1682
2714 https://github.com/STEllAR-GROUP/hpx/pull/1681
2715 https://github.com/STEllAR-GROUP/hpx/pull/1680
2716 https://github.com/STEllAR-GROUP/hpx/pull/1679
2717 https://github.com/STEllAR-GROUP/hpx/pull/1678
2718 https://github.com/STEllAR-GROUP/hpx/pull/1677
2719 https://github.com/STEllAR-GROUP/hpx/pull/1676
2720 https://github.com/STEllAR-GROUP/hpx/issues/1675
2721 https://github.com/STEllAR-GROUP/hpx/pull/1674
2722 https://github.com/STEllAR-GROUP/hpx/pull/1673
2723 https://github.com/STEllAR-GROUP/hpx/pull/1672
2724 https://github.com/STEllAR-GROUP/hpx/pull/1671
2725 https://github.com/STEllAR-GROUP/hpx/pull/1670
2726 https://github.com/STEllAR-GROUP/hpx/pull/1669
2727 https://github.com/STEllAR-GROUP/hpx/issues/1667
2728 https://github.com/STEllAR-GROUP/hpx/pull/1666
2729 https://github.com/STEllAR-GROUP/hpx/pull/1665
2730 https://github.com/STEllAR-GROUP/hpx/issues/1664
2731 https://github.com/STEllAR-GROUP/hpx/pull/1663
2732 https://github.com/STEllAR-GROUP/hpx/issues/1662
- PR #1661\(^{2733}\) - Added support for 128bit number serialization
- PR #1660\(^{2734}\) - Serialization 128bits
- PR #1659\(^{2735}\) - Implemented inner_product and adjacent_diff algos
- PR #1658\(^{2736}\) - Add serialization for std::set (as there is for std::vector and std::map)
- PR #1657\(^{2737}\) - Use of shared_ptr in io_service_pool changed to unique_ptr
- Issue #1656\(^{2738}\) - 1d_stencil codes all have wrong factor
- PR #1654\(^{2739}\) - When using runtime_mode_connect, find the correct localhost public ip address
- PR #1653\(^{2740}\) - Fixing 1617
- PR #1652\(^{2741}\) - Remove traits::action_may_require_id_splitting
- PR #1651\(^{2742}\) - Fixed performance counters related to AGAS cache timings
- PR #1650\(^{2743}\) - Remove leftovers of traits::type_size
- PR #1649\(^{2744}\) - Shorten target names on Windows to shorten used path names
- PR #1648\(^{2745}\) - Fixing problems introduced by merging #1623 for older compilers
- PR #1647\(^{2746}\) - Simplify running automatic builds on Windows
- Issue #1646\(^{2747}\) - Cache insert and update performance counters are broken
- Issue #1644\(^{2748}\) - Remove leftovers of traits::type_size
- Issue #1643\(^{2749}\) - Remove traits::action_may_require_id_splitting
- PR #1642\(^{2750}\) - Adds spell checker to the inspect tool for qbk and doxygen comments
- PR #1640\(^{2751}\) - First step towards fixing 688
- PR #1639\(^{2752}\) - Re-apply remaining changes from limit_dataflow_recursion branch
- PR #1638\(^{2753}\) - This fixes possible deadlock in the test ignore_while_locked_1485
- PR #1637\(^{2754}\) - Fixing hpx::wait_all() invoked with two vector<future<T>>
- PR #1636\(^{2755}\) - Partially re-apply changes from limit_dataflow_recursion branch

\(^{2733}\) https://github.com/STEllAR-GROUP/hpx/pull/1661
\(^{2734}\) https://github.com/STEllAR-GROUP/hpx/pull/1660
\(^{2735}\) https://github.com/STEllAR-GROUP/hpx/pull/1659
\(^{2736}\) https://github.com/STEllAR-GROUP/hpx/pull/1658
\(^{2737}\) https://github.com/STEllAR-GROUP/hpx/pull/1657
\(^{2738}\) https://github.com/STEllAR-GROUP/hpx/issues/1656
\(^{2739}\) https://github.com/STEllAR-GROUP/hpx/pull/1654
\(^{2740}\) https://github.com/STEllAR-GROUP/hpx/pull/1653
\(^{2741}\) https://github.com/STEllAR-GROUP/hpx/pull/1652
\(^{2742}\) https://github.com/STEllAR-GROUP/hpx/pull/1651
\(^{2743}\) https://github.com/STEllAR-GROUP/hpx/pull/1650
\(^{2744}\) https://github.com/STEllAR-GROUP/hpx/pull/1649
\(^{2745}\) https://github.com/STEllAR-GROUP/hpx/pull/1648
\(^{2746}\) https://github.com/STEllAR-GROUP/hpx/pull/1647
\(^{2747}\) https://github.com/STEllAR-GROUP/hpx/issues/1646
\(^{2748}\) https://github.com/STEllAR-GROUP/hpx/issues/1644
\(^{2749}\) https://github.com/STEllAR-GROUP/hpx/issues/1643
\(^{2750}\) https://github.com/STEllAR-GROUP/hpx/pull/1642
\(^{2751}\) https://github.com/STEllAR-GROUP/hpx/pull/1640
\(^{2752}\) https://github.com/STEllAR-GROUP/hpx/pull/1639
\(^{2753}\) https://github.com/STEllAR-GROUP/hpx/pull/1638
\(^{2754}\) https://github.com/STEllAR-GROUP/hpx/pull/1637
\(^{2755}\) https://github.com/STEllAR-GROUP/hpx/pull/1636
• PR #1635 - Adding missing test for #1572
• PR #1634 - Revert “Limit recursion-depth in dataflow to a configurable constant”
• PR #1633 - Add command line option to ignore batch environment
• PR #1631 - hpx::lcos::queue exhibits strange behavior
• PR #1630 - Fixed endline_whitespace_check.cpp to detect lines with only whitespace
• Issue #1629 - Inspect trailing whitespace checker problem
• PR #1628 - Removed meaningless const qualifiers. Minor icpc fix.
• PR #1627 - Fixing the queue LCO and add example demonstrating its use
• PR #1626 - Deprecating get_gid(), add get_id() and get_unmanaged_id()
• PR #1625 - Allowing to specify whether to send credits along with message
• Issue #1624 - Lifetime issue
• Issue #1623 - hpx::wait_all() invoked with two vector<future<T>> fails
• PR #1622 - Executor partitioners
• PR #1621 - Clean up coroutines implementation
• Issue #1620 - Revert #1535
• PR #1619 - Fix result type calculation for hpx::make_continuation
• PR #1618 - Fixing RDTSC on Xeon/Phi
• Issue #1617 - hpx cmake not working when run as a subproject
• Issue #1616 - cmake problem resulting in RDTSC not working correctly for Xeon Phi creates very strange results for duration counters
• Issue #1615 - hpx::make_continuation requires input and output to be the same
• PR #1614 - Fixed remove copy test
• Issue #1613 - Dataflow causes stack overflow
• PR #1612 - Modified foreach partitioner to use bulk execute

2756 https://github.com/STEllAR-GROUP/hpx/pull/1635
2757 https://github.com/STEllAR-GROUP/hpx/pull/1634
2758 https://github.com/STEllAR-GROUP/hpx/pull/1633
2759 https://github.com/STEllAR-GROUP/hpx/pull/1631
2760 https://github.com/STEllAR-GROUP/hpx/pull/1630
2761 https://github.com/STEllAR-GROUP/hpx/issues/1629
2762 https://github.com/STEllAR-GROUP/hpx/pull/1628
2763 https://github.com/STEllAR-GROUP/hpx/pull/1627
2764 https://github.com/STEllAR-GROUP/hpx/pull/1626
2765 https://github.com/STEllAR-GROUP/hpx/pull/1625
2766 https://github.com/STEllAR-GROUP/hpx/issues/1624
2767 https://github.com/STEllAR-GROUP/hpx/issues/1623
2768 https://github.com/STEllAR-GROUP/hpx/pull/1622
2769 https://github.com/STEllAR-GROUP/hpx/pull/1621
2770 https://github.com/STEllAR-GROUP/hpx/issues/1620
2771 https://github.com/STEllAR-GROUP/hpx/pull/1619
2772 https://github.com/STEllAR-GROUP/hpx/pull/1618
2773 https://github.com/STEllAR-GROUP/hpx/issues/1617
2774 https://github.com/STEllAR-GROUP/hpx/issues/1616
2775 https://github.com/STEllAR-GROUP/hpx/issues/1615
2776 https://github.com/STEllAR-GROUP/hpx/pull/1614
2777 https://github.com/STEllAR-GROUP/hpx/issues/1613
2778 https://github.com/STEllAR-GROUP/hpx/pull/1612
• PR #1611²⁷⁷⁹ - Limit recursion-depth in dataflow to a configurable constant
• PR #1610²⁷⁸⁰ - Increase timeout for CircleCI
• PR #1609²⁷⁸¹ - Refactoring thread manager, mainly extracting thread pool
• PR #1608²⁷⁸² - Fixed running multiple localities without localities parameter
• PR #1607²⁷⁸³ - More algorithm fixes to adjacentfind
• Issue #1606²⁷⁸⁴ - Running without localities parameter binds to bogus port range
• Issue #1605²⁷⁸⁵ - Too many serializations
• PR #1604²⁷⁸⁶ - Changes the HPX image into a hyperlink
• PR #1601²⁷⁸⁷ - Fixing problems with remove_copy algorithm tests
• PR #1600²⁷⁸⁸ - Actions with ids cleanup
• PR #1599²⁷⁸⁹ - Duplicate binding of global ids should fail
• PR #1598²⁷⁹⁰ - Fixing array access
• PR #1597²⁷⁹¹ - Improved the reliability of connecting/disconnecting localities
• Issue #1596²⁷⁹² - Duplicate id binding should fail
• PR #1595²⁷⁹³ - Fixing more cmake config constants
• PR #1594²⁷⁹⁴ - Fixing preprocessor constant used to enable C++11 chrono
• PR #1593²⁷⁹⁵ - Adding operator() for hpx::launch
• Issue #1592²⁷⁹⁶ - Error (typo) in the docs
• Issue #1590²⁷⁹⁷ - CMake fails when CMAKE_BINARY_DIR contains ‘+’.
• Issue #1589²⁷⁹⁸ - Disconnecting a locality results in segfault using heartbeat example
• PR #1588²⁷⁹⁹ - Fix doc string for config option HPX_WITH_EXAMPLES
• PR #1586²⁸⁰⁰ - Fixing 1493
• PR #1585²⁸⁰¹ - Additional Check for Inspect Tool to detect Endline Whitespace

²⁷⁷⁹ https://github.com/STEllAR-GROUP/hpx/pull/1611
²⁷⁸⁰ https://github.com/STEllAR-GROUP/hpx/pull/1610
²⁷⁸¹ https://github.com/STEllAR-GROUP/hpx/pull/1609
²⁷⁸² https://github.com/STEllAR-GROUP/hpx/pull/1608
²⁷⁸³ https://github.com/STEllAR-GROUP/hpx/pull/1607
²⁷⁸⁴ https://github.com/STEllAR-GROUP/hpx/issues/1606
²⁷⁸⁵ https://github.com/STEllAR-GROUP/hpx/issues/1605
²⁷⁸⁶ https://github.com/STEllAR-GROUP/hpx/pull/1604
²⁷⁸⁷ https://github.com/STEllAR-GROUP/hpx/pull/1601
²⁷⁸⁸ https://github.com/STEllAR-GROUP/hpx/pull/1600
²⁷⁸⁹ https://github.com/STEllAR-GROUP/hpx/pull/1599
²⁷⁹⁰ https://github.com/STEllAR-GROUP/hpx/pull/1598
²⁷⁹¹ https://github.com/STEllAR-GROUP/hpx/pull/1597
²⁷⁹² https://github.com/STEllAR-GROUP/hpx/issues/1596
²⁷⁹³ https://github.com/STEllAR-GROUP/hpx/pull/1595
²⁷⁹⁴ https://github.com/STEllAR-GROUP/hpx/pull/1594
²⁷⁹⁵ https://github.com/STEllAR-GROUP/hpx/pull/1593
²⁷⁹⁶ https://github.com/STEllAR-GROUP/hpx/pull/1592
²⁷⁹⁷ https://github.com/STEllAR-GROUP/hpx/issues/1590
²⁷⁹⁸ https://github.com/STEllAR-GROUP/hpx/issues/1589
²⁷⁹⁹ https://github.com/STEllAR-GROUP/hpx/pull/1588
²⁸⁰⁰ https://github.com/STEllAR-GROUP/hpx/pull/1586
²⁸⁰¹ https://github.com/STEllAR-GROUP/hpx/pull/1585
• Issue #1584  - Clean up coroutines implementation
• PR #1583  - Adding a check for end line whitespace
• PR #1582  - Attempt to fix assert firing after scheduling loop was exited
• PR #1581  - Fixed adjacentfind_binary test
• PR #1580  - Prevent some of the internal cmake lists from growing indefinitely
• PR #1579  - Removing type_size trait, replacing it with special archive type
• Issue #1578  - Remove demangle_helper
• PR #1577  - Get ptr problems
• Issue #1576  - Refactor async, dataflow, and future::then
• PR #1575  - Fixing tests for parallel rotate
• PR #1574  - Cleaning up schedulers
• PR #1573  - Fixing thread pool executor
• PR #1572  - Fixing number of configured localities
• PR #1571  - Reimplement decay
• PR #1570  - Refactoring async, apply, and dataflow APIs
• PR #1569  - Changed range for mach-o library lookup
• PR #1568  - Mark decltype support as required
• PR #1567  - Removed const from algorithms
• Issue #1566  - CMAKE Configuration Test Failures for clang 3.5 on debian
• PR #1565  - Dylib support
• PR #1564  - Converted partitioners and some algorithms to use executors
• PR #1563  - Fix several #includes for Boost.Preprocessor
• PR #1562  - Adding configuration option disabling/enabling all message handlers

https://github.com/STEllAR-GROUP/hpx/issues/1584
https://github.com/STEllAR-GROUP/hpx/pull/1583
https://github.com/STEllAR-GROUP/hpx/pull/1582
https://github.com/STEllAR-GROUP/hpx/pull/1581
https://github.com/STEllAR-GROUP/hpx/pull/1580
https://github.com/STEllAR-GROUP/hpx/pull/1579
https://github.com/STEllAR-GROUP/hpx/pull/1578
https://github.com/STEllAR-GROUP/hpx/pull/1577
https://github.com/STEllAR-GROUP/hpx/pull/1576
https://github.com/STEllAR-GROUP/hpx/pull/1575
https://github.com/STEllAR-GROUP/hpx/pull/1574
https://github.com/STEllAR-GROUP/hpx/pull/1573
https://github.com/STEllAR-GROUP/hpx/pull/1572
https://github.com/STEllAR-GROUP/hpx/pull/1571
https://github.com/STEllAR-GROUP/hpx/pull/1570
https://github.com/STEllAR-GROUP/hpx/pull/1569
https://github.com/STEllAR-GROUP/hpx/pull/1568
https://github.com/STEllAR-GROUP/hpx/pull/1567
https://github.com/STEllAR-GROUP/hpx/pull/1566
https://github.com/STEllAR-GROUP/hpx/pull/1565
https://github.com/STEllAR-GROUP/hpx/pull/1564
https://github.com/STEllAR-GROUP/hpx/pull/1563
https://github.com/STEllAR-GROUP/hpx/pull/1562
• PR #1561\textsuperscript{2825} - Removed all occurrences of boost::move replacing it with std::move
• Issue #1560\textsuperscript{2826} - Leftover HPX_REGISTER_ACTION_DECLARATION_2
• PR #1558\textsuperscript{2827} - Revisit async/apply SFINAE conditions
• PR #1557\textsuperscript{2828} - Removing type_size trait, replacing it with special archive type
• PR #1556\textsuperscript{2829} - Executor algorithms
• PR #1555\textsuperscript{2830} - Remove the necessity to specify archive flags on the receiving end
• PR #1554\textsuperscript{2831} - Removing obsolete Boost.Serialization macros
• PR #1553\textsuperscript{2832} - Properly fix HPX_DEFINE_*_ACTION macros
• PR #1552\textsuperscript{2833} - Fixed algorithms relying on copy_if implementation
• PR #1551\textsuperscript{2834} - Pxfs - Modifying FindOrangeFS.cmake based on OrangeFS 2.9.X
• Issue #1550\textsuperscript{2835} - Passing plain identifier inside HPX_DEFINE_PLAIN_ACTION_1
• PR #1549\textsuperscript{2836} - Fixing intel14/libstdc++4.4
• PR #1548\textsuperscript{2837} - Moving raw_ptr to detail namespace
• PR #1547\textsuperscript{2838} - Adding support for executors to future.then
• PR #1546\textsuperscript{2839} - Executor traits result types
• PR #1545\textsuperscript{2840} - Integrate executors with dataflow
• PR #1543\textsuperscript{2841} - Fix potential zero-copy for primarynamespace::bulk_service_async et.al.
• PR #1542\textsuperscript{2842} - Merging HPX0.9.10 into pxfs branch
• PR #1541\textsuperscript{2843} - Removed HPX0.9.10 into pxfs branch
• PR #1540\textsuperscript{2844} - Fix idle-rate on platforms without TSC
• PR #1539\textsuperscript{2845} - Reporting situation if zero-copy-serialization was performed by a parcel generated from a plain apply/async
• PR #1538\textsuperscript{2846} - Changed return type of bulk executors and added test
• Issue #1537\textsuperscript{2847} - Incorrect cpuid config tests

\textsuperscript{2825} https://github.com/STEllAR-GROUP/hpx/pull/1561
\textsuperscript{2826} https://github.com/STEllAR-GROUP/hpx/issues/1560
\textsuperscript{2827} https://github.com/STEllAR-GROUP/hpx/pull/1558
\textsuperscript{2828} https://github.com/STEllAR-GROUP/hpx/pull/1557
\textsuperscript{2829} https://github.com/STEllAR-GROUP/hpx/pull/1556
\textsuperscript{2830} https://github.com/STEllAR-GROUP/hpx/pull/1555
\textsuperscript{2831} https://github.com/STEllAR-GROUP/hpx/pull/1554
\textsuperscript{2832} https://github.com/STEllAR-GROUP/hpx/pull/1553
\textsuperscript{2833} https://github.com/STEllAR-GROUP/hpx/pull/1552
\textsuperscript{2834} https://github.com/STEllAR-GROUP/hpx/pull/1551
\textsuperscript{2835} https://github.com/STEllAR-GROUP/hpx/issues/1550
\textsuperscript{2836} https://github.com/STEllAR-GROUP/hpx/pull/1549
\textsuperscript{2837} https://github.com/STEllAR-GROUP/hpx/pull/1548
\textsuperscript{2838} https://github.com/STEllAR-GROUP/hpx/pull/1547
\textsuperscript{2839} https://github.com/STEllAR-GROUP/hpx/pull/1546
\textsuperscript{2840} https://github.com/STEllAR-GROUP/hpx/pull/1545
\textsuperscript{2841} https://github.com/STEllAR-GROUP/hpx/pull/1543
\textsuperscript{2842} https://github.com/STEllAR-GROUP/hpx/pull/1542
\textsuperscript{2843} https://github.com/STEllAR-GROUP/hpx/pull/1541
\textsuperscript{2844} https://github.com/STEllAR-GROUP/hpx/pull/1540
\textsuperscript{2845} https://github.com/STEllAR-GROUP/hpx/pull/1539
\textsuperscript{2846} https://github.com/STEllAR-GROUP/hpx/pull/1538
\textsuperscript{2847} https://github.com/STEllAR-GROUP/hpx/issues/1537
• PR #1536 - Changed return type of bulk executors and added test
• PR #1535 - Make sure promise::get_gid() can be called more than once
• PR #1534 - Fixed async_callback with bound callback
• PR #1533 - Updated the link in the documentation to a publically-accessible URL
• PR #1532 - Make sure sync primitives are not copyable nor movable
• PR #1531 - Fix unwrapped issue with future ranges of void type
• PR #1530 - Serialization complex
• Issue #1528 - Unwrapped issue with future<void>
• Issue #1527 - HPX does not build with Boost 1.58.0
• PR #1526 - Added support for boost.multi_array serialization
• PR #1525 - Properly handle deferred futures, fixes #1506
• PR #1524 - Making sure invalid action argument types generate clear error message
• Issue #1523 - Need serialization support for boost multi array
• Issue #1521 - Remote async and zero-copy serialization optimizations don’t play well together
• PR #1520 - Fixing UB while registering polymorphic classes for serialization
• PR #1519 - Making detail::condition_variable safe to use
• PR #1518 - Fix when_some bug missing indices in its result
• Issue #1517 - Typo may affect CMake build system tests
• PR #1516 - Fixing Posix context
• PR #1515 - Fixing Posix context
• PR #1514 - Correct problems with loading dynamic components
• PR #1513 - Fixing intel glibc4 4
• Issue #1508 - memory and papi counters do not work

2848 https://github.com/STEllAR-GROUP/hpx/pull/1536
2849 https://github.com/STEllAR-GROUP/hpx/pull/1535
2850 https://github.com/STEllAR-GROUP/hpx/pull/1534
2851 https://github.com/STEllAR-GROUP/hpx/pull/1533
2852 https://github.com/STEllAR-GROUP/hpx/pull/1532
2853 https://github.com/STEllAR-GROUP/hpx/pull/1531
2854 https://github.com/STEllAR-GROUP/hpx/pull/1530
2855 https://github.com/STEllAR-GROUP/hpx/issues/1528
2856 https://github.com/STEllAR-GROUP/hpx/issues/1527
2857 https://github.com/STEllAR-GROUP/hpx/pull/1526
2858 https://github.com/STEllAR-GROUP/hpx/pull/1525
2859 https://github.com/STEllAR-GROUP/hpx/pull/1524
2860 https://github.com/STEllAR-GROUP/hpx/issues/1522
2861 https://github.com/STEllAR-GROUP/hpx/issues/1521
2862 https://github.com/STEllAR-GROUP/hpx/pull/1520
2863 https://github.com/STEllAR-GROUP/hpx/pull/1519
2864 https://github.com/STEllAR-GROUP/hpx/pull/1518
2865 https://github.com/STEllAR-GROUP/hpx/issues/1517
2866 https://github.com/STEllAR-GROUP/hpx/pull/1516
2867 https://github.com/STEllAR-GROUP/hpx/pull/1515
2868 https://github.com/STEllAR-GROUP/hpx/pull/1514
2869 https://github.com/STEllAR-GROUP/hpx/pull/1513
2870 https://github.com/STEllAR-GROUP/hpx/issues/1508
• Issue #1507 - Unrecognized Command Line Option Error causing exit status 0
• Issue #1506 - Properly handle deferred futures
• PR #1505 - Adding #include - would not compile without this
• Issue #1502 - boost::filesystem::exists throws unexpected exception
• Issue #1501 - hwloc configuration options are wrong for MIC
• PR #1504 - Making sure boost::filesystem::exists() does not throw
• PR #1500 - Exit application on --hpx:version/-v and --hpx:info
• PR #1498 - Extended task block
• PR #1497 - Unique ptr serialization
• PR #1496 - Unique ptr serialization (closed)
• PR #1495 - Switching circleci build type to debug
• Issue #1494 - --hpx:version/-v does not exit after printing version information
• Issue #1493 - add an hpx_ prefix to libraries and components to avoid name conflicts
• Issue #1492 - Define and ensure limitations for arguments to async/apply
• PR #1489 - Enable idle rate counter on demand
• PR #1488 - Made sure detail::condition_variable can be safely destroyed
• PR #1487 - Introduced default (main) template implementation for ignore_while_checking
• PR #1486 - Add HPX inspect tool
• Issue #1485 - ignore_while_locked doesn’t support all Lockable types
• PR #1484 - Docker image generation
• PR #1483 - Move external endian library into HPX
• PR #1482 - Actions with integer type ids
• Issue #1481 - Sync primitives safe destruction

https://github.com/STEllAR-GROUP/hpx/issues/1507
https://github.com/STEllAR-GROUP/hpx/issues/1506
https://github.com/STEllAR-GROUP/hpx/pull/1505
https://github.com/STEllAR-GROUP/hpx/pull/1502
https://github.com/STEllAR-GROUP/hpx/pull/1501
https://github.com/STEllAR-GROUP/hpx/pull/1504
https://github.com/STEllAR-GROUP/hpx/pull/1500
https://github.com/STEllAR-GROUP/hpx/pull/1498
https://github.com/STEllAR-GROUP/hpx/pull/1497
https://github.com/STEllAR-GROUP/hpx/pull/1496
https://github.com/STEllAR-GROUP/hpx/pull/1495
https://github.com/STEllAR-GROUP/hpx/pull/1494
https://github.com/STEllAR-GROUP/hpx/pull/1493
https://github.com/STEllAR-GROUP/hpx/pull/1492
https://github.com/STEllAR-GROUP/hpx/pull/1489
https://github.com/STEllAR-GROUP/hpx/pull/1488
https://github.com/STEllAR-GROUP/hpx/pull/1487
https://github.com/STEllAR-GROUP/hpx/pull/1486
https://github.com/STEllAR-GROUP/hpx/pull/1485
https://github.com/STEllAR-GROUP/hpx/pull/1484
https://github.com/STEllAR-GROUP/hpx/pull/1483
https://github.com/STEllAR-GROUP/hpx/pull/1482
https://github.com/STEllAR-GROUP/hpx/pull/1481
• Issue #1480 - Move external/boost/endian into hpx/util
• Issue #1478 - Boost inspect violations
• PR #1479 - Adds serialization for arrays; some futher/minor fixes
• PR #1477 - Fixing problems with the Intel compiler using a GCC 4.4 std library
• PR #1478 - Adding hpx::lcos::latch and hpx::lcos::local::latch
• Issue #1475 - Boost inspect violations
• PR #1473 - Fixing action move tests
• Issue #1471 - Sync primitives should not be movable
• PR #1476 - Adding hpx::lcos::latch and hpx::lcos::local::latch
• Issue #1470 - Fixing problems with the Intel compiler using a GCC 4.4 std library
• PR #1468 - Fixed container creation
• Issue #1467 - HPX application fail during finalization
• Issue #1466 - HPX doesn’t pick up Torque’s nodefile on SuperMIC
• Issue #1464 - HPX option for pre and post bootstrap performance counters
• PR #1463 - Replacing async_colocated(id, ...) with async(colocated(id), ...)
• PR #1462 - Consolidated task_region with N4411
• PR #1461 - Consolidate inconsistent CMake option names
• Issue #1460 - Which malloc is actually used? or at least which one is HPX built with
• Issue #1459 - Make cmake configure step fail explicitly if compiler version is not supported
• Issue #1458 - Update parallel::task_region with N4411
• PR #1456 - Consolidating new_<>()
• Issue #1455 - Replace async_colocated(id, ...) with async(colocated(id), ...)
• PR #1454 - Removed harmful std::moves from return statements
• PR #1453 - Use range-based for-loop instead of Boost.Foreach

https://github.com/STEllAR-GROUP/hpx/issues/1480
https://github.com/STEllAR-GROUP/hpx/issues/1478
https://github.com/STEllAR-GROUP/hpx/pull/1479
https://github.com/STEllAR-GROUP/hpx/pull/1477
https://github.com/STEllAR-GROUP/hpx/pull/1476
https://github.com/STEllAR-GROUP/hpx/issues/1475
https://github.com/STEllAR-GROUP/hpx/pull/1473
https://github.com/STEllAR-GROUP/hpx/issues/1471
https://github.com/STEllAR-GROUP/hpx/pull/1470
https://github.com/STEllAR-GROUP/hpx/pull/1468
https://github.com/STEllAR-GROUP/hpx/issues/1467
https://github.com/STEllAR-GROUP/hpx/issues/1466
https://github.com/STEllAR-GROUP/hpx/pull/1464
https://github.com/STEllAR-GROUP/hpx/pull/1463
https://github.com/STEllAR-GROUP/hpx/pull/1462
https://github.com/STEllAR-GROUP/hpx/pull/1461
https://github.com/STEllAR-GROUP/hpx/issues/1460
https://github.com/STEllAR-GROUP/hpx/issues/1459
https://github.com/STEllAR-GROUP/hpx/issues/1458
https://github.com/STEllAR-GROUP/hpx/pull/1456
https://github.com/STEllAR-GROUP/hpx/issues/1455
https://github.com/STEllAR-GROUP/hpx/pull/1454
https://github.com/STEllAR-GROUP/hpx/pull/1453
- PR #1452 - C++ feature tests
- PR #1451 - When serializing, pass archive flags to traits::get_type_size
- Issue #1450 - traits::get_type_size needs archive flags to enable zero_copy optimizations
- Issue #1449 - “couldn’t create performance counter” - AGAS
- Issue #1448 - Replace distributing factories with new_<T[]>(...)
- PR #1447 - Removing obsolete remote_object component
- PR #1446 - Hpx serialization
- PR #1445 - Removing obsolete remote_object component
- PR #1444 - Replacing travis with circleci
- PR #1443 - Always stripping HPX command line arguments before executing start function
- PR #1442 - Adding --hpx:bind=none to disable thread affinities
- Issue #1439 - Libraries get linked in multiple times, RPATH is not properly set
- PR #1438 - Removed superfluous typedefs
- Issue #1437 - hpx::init() should strip HPX-related flags from argv
- Issue #1436 - Add strong scaling option to hpx
- PR #1435 - Adding async_cb, async_continue_cb, and async_colocated_cb
- PR #1434 - Added missing install rule, removed some dead CMake code
- PR #1433 - Add GitExternal and SubProject cmake scripts from eyescale/cmake repo
- Issue #1432 - Add command line flag to disable thread pinning
- PR #1431 - Fix #1423
- Issue #1430 - Inconsistent CMake option names
- Issue #1429 - Configure setting HPX_HAVE_PARCELPORT_MPI is ignored
- PR #1428 - Fixes #1419 (closed)
- PR #1427 - Adding stencil_iterator and transform_iterator

https://github.com/STEllAR-GROUP/hpx/pull/1452
https://github.com/STEllAR-GROUP/hpx/pull/1451
https://github.com/STEllAR-GROUP/hpx/issues/1450
https://github.com/STEllAR-GROUP/hpx/issues/1449
https://github.com/STEllAR-GROUP/hpx/issues/1448
https://github.com/STEllAR-GROUP/hpx/pull/1447
https://github.com/STEllAR-GROUP/hpx/pull/1446
https://github.com/STEllAR-GROUP/hpx/pull/1445
https://github.com/STEllAR-GROUP/hpx/pull/1443
https://github.com/STEllAR-GROUP/hpx/pull/1442
https://github.com/STEllAR-GROUP/hpx/issues/1439
https://github.com/STEllAR-GROUP/hpx/pull/1438
https://github.com/STEllAR-GROUP/hpx/issues/1437
https://github.com/STEllAR-GROUP/hpx/issues/1436
https://github.com/STEllAR-GROUP/hpx/pull/1435
https://github.com/STEllAR-GROUP/hpx/pull/1434
https://github.com/STEllAR-GROUP/hpx/pull/1433
https://github.com/STEllAR-GROUP/hpx/pull/1432
https://github.com/STEllAR-GROUP/hpx/pull/1431
https://github.com/STEllAR-GROUP/hpx/pull/1430
https://github.com/STEllAR-GROUP/hpx/issues/1429
https://github.com/STEllAR-GROUP/hpx/pull/1428
https://github.com/STEllAR-GROUP/hpx/pull/1427
• PR #1426 - Fixes #1419
• PR #1425 - During serialization memory allocation should honour allocator chunk size
• Issue #1424 - chunk allocation during serialization does not use memory pool/allocator chunk size
• Issue #1423 - Remove HPX_STD_UNIQUE_PTR
• Issue #1422 - hpx:threads=all allocates too many os threads
• PR #1420 - added .travis.yml
• Issue #1419 - Unify enums: hpx::runtime::state and hpx::state
• PR #1416 - Adding travis builder
• Issue #1414 - Correct directory for dispatch_gcc46.hpp iteration
• Issue #1410 - Set operation algorithms
• Issue #1389 - Parallel algorithms relying on scan partitioner break for small number of elements
• Issue #1325 - Exceptions thrown during parcel handling are not handled correctly
• Issue #1315 - Errors while running performance tests
• Issue #1309 - hpx::vector partitions are not easily extendable by applications
• PR #1300 - Added serialization/de-serialization to examples.tuplespace
• Issue #1251 - hpx:threads::get_thread_count doesn’t consider pending threads
• Issue #1008 - Decrease in application performance overtime; occasional spikes of major slowdown
• Issue #1001 - Zero copy serialization raises assert
• Issue #721 - Make HPX usable for Xeon Phi
• Issue #524 - Extend scheduler to support threads which can’t be stolen

2.11.10 HPX V0.9.10 (Mar 24, 2015)

General changes

This is the 12th official release of HPX. It coincides with the 7th anniversary of the first commit to our source code repository. Since then, we have seen over 12300 commits amounting to more than 220000 lines of C++ code.
The major focus of this release was to improve the reliability of large scale runs. We believe to have achieved this goal as we now can reliably run HPX applications on up to ~24k cores. We have also shown that HPX can be used with success for symmetric runs (applications using both, host cores and Intel Xeon/Phi coprocessors). This is a huge step forward in terms of the usability of HPX. The main focus of this work involved isolating the causes of the segmentation faults at start up and shut down. Many of these issues were discovered to be the result of the suspension of threads which hold locks.

A very important improvement introduced with this release is the refactoring of the code representing our parcel-port implementation. Parcel-ports can now be implemented by 3rd parties as independent plugins which are dynamically loaded at runtime (static linking of parcel-ports is also supported). This refactoring also includes a massive improvement of the performance of our existing parcel-ports. We were able to significantly reduce the networking latencies and to improve the available networking bandwidth. Please note that in this release we disabled the ibverbs and ipc parcel ports as those have not been ported to the new plugin system yet (see Issue #839).

Another corner stone of this release is our work towards a complete implementation of __cpp11_n4104__ (Working Draft, Technical Specification for C++ Extensions for Parallelism). This document defines a set of parallel algorithms to be added to the C++ standard library. We now have implemented about 75% of all specified parallel algorithms (see [link hpx.manual.parallel.parallel_algorithms Parallel Algorithms] for more details). We also implemented some extensions to __cpp11_n4104__ allowing to invoke all of the algorithms asynchronously.

This release adds a first implementation of hpx::vector which is a distributed data structure closely aligned to the functionality of std::vector. The difference is that hpx::vector stores the data in partitions where the partitions can be distributed over different localities. We started to work on allowing to use the parallel algorithms with hpx::vector. At this point we have implemented only a few of the parallel algorithms to support distributed data structures (like hpx::vector) for testing purposes (see Issue #1338 for a documentation of our progress).

Breaking changes

With this release we put a lot of effort into changing the code base to be more compatible to C++11. These changes have caused the following issues for backward compatibility:

- **Move to Variadics**- All of the API now uses variadic templates. However, this change required to modify the argument sequence for some of the exiting API functions (hpx::async_continue, hpx::apply_continue, hpx::when_each, hpx::wait_each, synchronous invocation of actions).

- **Changes to Macros**- We also removed the macros HPX_STD_FUNCTION and HPX_STD_TUPLE. This shouldn’t affect any user code as we replaced HPX_STD_FUNCTION with hpx::util::function_nonser which was the default expansion used for this macro. All HPX API functions which expect a hpx::util::function_nonser (or a hpx::util::unique_function_nonser) can now be transparently called with a compatible std::function instead. Similarly, HPX_STD_TUPLE was replaced by its default expansion as well: hpx::util::tuple.

- **Changes to hpx::unique_future-** hpx::unique_future, which was deprecated in the previous release for hpx::future is now completely removed from HPX. This completes the transition to a completely standards conforming implementation of hpx::future.

- **Changes to Supported Compilers.** Finally, in order to utilize more C++11 semantics, we have officially dropped support for GCC 4.4 and MSVC 2012. Please see our Prerequisites page for more details.

Bug fixes (closed tickets)

Here is a list of the important tickets we closed for this release.

- Issue #1402 Internal shared_future serialization copies

---

2.11. Releases

997
• Issue #1399 - Build takes unusually long time...
• Issue #1398 - Tests using the scan partitioner are broken on at least gcc 4.7 and intel compiler
• Issue #1397 - Completely remove hpx::unique_future
• Issue #1396 - Parallel scan algorithms with different initial values
• Issue #1395 - Race Condition - 1d_stencil_8 - SuperMIC
• Issue #1394 - "suspending thread while at least one lock is being held" - 1d_stencil_8 - SuperMIC
• Issue #1393 - SEGFAULT in 1d_stencil_8 on SuperMIC
• Issue #1392 - Fixing #1168
• Issue #1391 - Parallel Algorithms for scan partitioner for small number of elements
• Issue #1387 - Failure with more than 4 localities
• Issue #1386 - Dispatching unhandled exceptions to outer user code
• Issue #1385 - Adding Copy algorithms, fixing parallel::copy_if
• Issue #1384 - Fixing 1325
• Issue #1383 - Fixed #504: Refactor Dataflow LCO to work with futures, this removes the dataflow component as it is obsolete
• Issue #1382 - is_sorted, is_sorted_until and is_partitioned algorithms
• Issue #1381 - fix for CMake versions prior to 3.1
• Issue #1380 - resolved warning in CMake 3.1 and newer
• Issue #1379 - Compilation error with papi
• Issue #1378 - Towards safer migration
• Issue #1377 - HPXConfig.cmake should include TCMALLOC_LIBRARY and TCMALLOC_INCLUDE_DIR
• Issue #1376 - Warning on uninitialized member
• Issue #1375 - Fixing 1163
• Issue #1374 - Fixing the MSVC 12 release builder
• Issue #1373 - Modifying parallel search algorithm for zero length searches
• Issue #1372 - Modifying parallel search algorithm for zero length searches
• Issue #1371 - Avoid holding a lock during agas::inref while doing a credit split
• Issue #1370 - --hpx:bind throws unexpected error
• Issue #1369 - Getting rid of (void) in loops
• Issue #1368 - Variadic templates support for tuple
• Issue #1367 - One last batch of variadic templates support
• Issue #1366 - Fixing symbolic namespace hang
• Issue #1365 - More held locks
• Issue #1364 - Add counters 1363
• Issue #1363 - Add thread overhead counters
• Issue #1362 - Std config removal
• Issue #1361 - Parcelport plugins
• Issue #1360 - Detuplify transfer_action
• Issue #1359 - Removed obsolete checks
• Issue #1358 - Fixing 1352
• Issue #1357 - Variadic templates support for runtime_support and components
• Issue #1356 - fixed coordinate test for intel13
• Issue #1355 - fixed coordinate.hpp
• Issue #1354 - Lexicographical Compare completed
• Issue #1353 - HPX should set Boost_ADDITIONAL_VERSIONS flags
• Issue #1352 - Error: Cannot find action "" in type registry: HPX(bad_action_code)
• Issue #1351 - Variadic templates support for appliers

https://github.com/STEllAR-GROUP/hpx/issues/1373
https://github.com/STEllAR-GROUP/hpx/issues/1372
https://github.com/STEllAR-GROUP/hpx/issues/1371
https://github.com/STEllAR-GROUP/hpx/issues/1370
https://github.com/STEllAR-GROUP/hpx/issues/1369
https://github.com/STEllAR-GROUP/hpx/issues/1368
https://github.com/STEllAR-GROUP/hpx/issues/1367
https://github.com/STEllAR-GROUP/hpx/issues/1366
https://github.com/STEllAR-GROUP/hpx/issues/1365
https://github.com/STEllAR-GROUP/hpx/issues/1364
https://github.com/STEllAR-GROUP/hpx/issues/1363
https://github.com/STEllAR-GROUP/hpx/issues/1362
https://github.com/STEllAR-GROUP/hpx/issues/1361
https://github.com/STEllAR-GROUP/hpx/issues/1360
https://github.com/STEllAR-GROUP/hpx/issues/1359
https://github.com/STEllAR-GROUP/hpx/issues/1358
https://github.com/STEllAR-GROUP/hpx/issues/1357
https://github.com/STEllAR-GROUP/hpx/issues/1356
https://github.com/STEllAR-GROUP/hpx/issues/1355
https://github.com/STEllAR-GROUP/hpx/issues/1354
https://github.com/STEllAR-GROUP/hpx/issues/1353
https://github.com/STEllAR-GROUP/hpx/issues/1352
https://github.com/STEllAR-GROUP/hpx/issues/1351
HPX Documentation, 1.4.1

- Issue #1350\(^{3009}\) - Actions simplification
- Issue #1349\(^{3010}\) - Variadic when and wait functions
- Issue #1348\(^{3011}\) - Added hpx_init header to test files
- Issue #1347\(^{3012}\) - Another batch of variadic templates support
- Issue #1346\(^{3013}\) - Segmented copy
- Issue #1345\(^{3014}\) - Attempting to fix hangs during shutdown
- Issue #1344\(^{3015}\) - Std config removal
- Issue #1343\(^{3016}\) - Removing various distribution policies for hpx::vector
- Issue #1342\(^{3017}\) - Inclusive scan
- Issue #1341\(^{3018}\) - Exclusive scan
- Issue #1340\(^{3019}\) - Adding parallel::count for distributed data structures, adding tests
- Issue #1339\(^{3020}\) - Update argument order for transform_reduce
- Issue #1337\(^{3021}\) - Fix dataflow to handle properly ranges of futures
- Issue #1336\(^{3022}\) - dataflow needs to hold onto futures passed to it
- Issue #1335\(^{3023}\) - Fails to compile with msvc14
- Issue #1334\(^{3024}\) - Examples build problem
- Issue #1333\(^{3025}\) - Distributed transform reduce
- Issue #1332\(^{3026}\) - Variadic templates support for actions
- Issue #1331\(^{3027}\) - Some ambiguous calls of map::erase have been prevented by adding additional check in locality constructor.
- Issue #1330\(^{3028}\) - Defining Plain Actions does not work as described in the documentation
- Issue #1329\(^{3029}\) - Distributed vector cleanup
- Issue #1328\(^{3030}\) - Sync docs and comments with code in hello_world example
- Issue #1327\(^{3031}\) - Typos in docs

\(^{3009}\) https://github.com/STEllAR-GROUP/hpx/issues/1350
^{3010}\) https://github.com/STEllAR-GROUP/hpx/issues/1349
^{3011}\) https://github.com/STEllAR-GROUP/hpx/issues/1348
^{3012}\) https://github.com/STEllAR-GROUP/hpx/issues/1347
^{3013}\) https://github.com/STEllAR-GROUP/hpx/issues/1346
^{3014}\) https://github.com/STEllAR-GROUP/hpx/issues/1345
^{3015}\) https://github.com/STEllAR-GROUP/hpx/issues/1344
^{3016}\) https://github.com/STEllAR-GROUP/hpx/issues/1343
^{3017}\) https://github.com/STEllAR-GROUP/hpx/issues/1342
^{3018}\) https://github.com/STEllAR-GROUP/hpx/issues/1341
^{3019}\) https://github.com/STEllAR-GROUP/hpx/issues/1340
^{3020}\) https://github.com/STEllAR-GROUP/hpx/issues/1339
^{3021}\) https://github.com/STEllAR-GROUP/hpx/issues/1337
^{3022}\) https://github.com/STEllAR-GROUP/hpx/issues/1336
^{3023}\) https://github.com/STEllAR-GROUP/hpx/issues/1335
^{3024}\) https://github.com/STEllAR-GROUP/hpx/issues/1334
^{3025}\) https://github.com/STEllAR-GROUP/hpx/issues/1333
^{3026}\) https://github.com/STEllAR-GROUP/hpx/issues/1332
^{3027}\) https://github.com/STEllAR-GROUP/hpx/issues/1331
^{3028}\) https://github.com/STEllAR-GROUP/hpx/issues/1330
^{3029}\) https://github.com/STEllAR-GROUP/hpx/issues/1329
^{3030}\) https://github.com/STEllAR-GROUP/hpx/issues/1328
^{3031}\) https://github.com/STEllAR-GROUP/hpx/issues/1327
• Issue #1326 - Documentation and code diverged in Fibonacci tutorial
• Issue #1325 - Exceptions thrown during parcel handling are not handled correctly
• Issue #1324 - fixed bandwidth calculation
• Issue #1323 - mmap() failed to allocate thread stack due to insufficient resources
• Issue #1322 - HPX fails to build aa182cf
• Issue #1321 - Limiting size of outgoing messages while coalescing parcels
• Issue #1320 - passing a future with launch::deferred in remote function call causes hang
• Issue #1319 - An exception when tries to specify number high priority threads with abp-priority
• Issue #1318 - Unable to run program with abp-priority and numa-sensitivity enabled
• Issue #1317 - N4071 Search/Search_n finished, minor changes
• Issue #1316 - Add config option to make -lhpx.run_hpx_main!=1 the default
• Issue #1314 - Variadic support for async and apply
• Issue #1313 - Adjust when_any/some to the latest proposed interfaces
• Issue #1312 - Fixing #857: hpx::naming::locality leaks parcelport specific information into the public interface
• Issue #1311 - Distributed get’er/set’er_values for distributed vector
• Issue #1310 - Crashing in hpx::parcelset::policies::mpi::connection_handler::handle_messages() on SuperMIC
• Issue #1308 - Unable to execute an application with –hpx:threads
• Issue #1307 - merge_graph linking issue
• Issue #1306 - First batch of variadic templates support
• Issue #1305 - Create a compiler wrapper
• Issue #1304 - Provide a compiler wrapper for hpx
• Issue #1303 - Drop support for GCC44

3032 https://github.com/STEllAR-GROUP/hpx/issues/1326
3033 https://github.com/STEllAR-GROUP/hpx/issues/1325
3034 https://github.com/STEllAR-GROUP/hpx/issues/1324
3035 https://github.com/STEllAR-GROUP/hpx/issues/1323
3036 https://github.com/STEllAR-GROUP/hpx/issues/1322
3037 https://github.com/STEllAR-GROUP/hpx/issues/1321
3038 https://github.com/STEllAR-GROUP/hpx/issues/1319
3039 https://github.com/STEllAR-GROUP/hpx/issues/1318
3040 https://github.com/STEllAR-GROUP/hpx/issues/1317
3041 https://github.com/STEllAR-GROUP/hpx/issues/1316
3042 https://github.com/STEllAR-GROUP/hpx/issues/1314
3043 https://github.com/STEllAR-GROUP/hpx/issues/1313
3044 https://github.com/STEllAR-GROUP/hpx/issues/1312
3045 https://github.com/STEllAR-GROUP/hpx/issues/1311
3046 https://github.com/STEllAR-GROUP/hpx/issues/1308
3047 https://github.com/STEllAR-GROUP/hpx/issues/1307
3048 https://github.com/STEllAR-GROUP/hpx/issues/1306
3049 https://github.com/STEllAR-GROUP/hpx/issues/1305
3050 https://github.com/STEllAR-GROUP/hpx/issues/1304
3051 https://github.com/STEllAR-GROUP/hpx/issues/1303
• Issue #1302 - Fixing #1297
• Issue #1301 - Compilation error when tried to use boost range iterators with wait_all
• Issue #1298 - Distributed vector
• Issue #1297 - Unable to invoke component actions recursively
• Issue #1294 - HDF5 build error
• Issue #1275 - The parcelport implementation is non-optimal
• Issue #1267 - Added classes and unit tests for local_file, orangefs_file and pxfs_file
• Issue #1264 - Error “assertion ‘!m_fun’ failed” randomly occurs when using TCP
• Issue #1254 - thread binding seems to not work properly
• Issue #1220 - parallel::copy_if is broken
• Issue #1217 - Find a better way of fixing the issue patched by #1216
• Issue #1168 - Starting HPX on Cray machines using aprun isn’t working correctly
• Issue #1085 - Replace startup and shutdown barriers with broadcasts
• Issue #981 - With SLURM, –hpx:threads=8 should not be necessary
• Issue #857 - hpx::naming::locality leaks parcelport specific information into the public interface
• Issue #850 - “flush” not documented
• Issue #763 - Create buildbot instance that uses std::bind as HPX_STD_BIND
• Issue #680 - Convert parcel ports into a plugin system
• Issue #582 - Make exception thrown from HPX threads available from hpx::init
• Issue #504 - Refactor Dataflow LCO to work with futures
• Issue #196 - Don’t store copies of the locality network metadata in the gva table
2.11.11 HPX V0.9.9 (Oct 31, 2014, codename Spooky)

General changes

We have had over 1500 commits since the last release and we have closed over 200 tickets (bugs, feature requests, pull requests, etc.). These are by far the largest numbers of commits and resolved issues for any of the HPX releases so far. We are especially happy about the large number of people who contributed for the first time to HPX.

• We completed the transition from the older (non-conforming) implementation of `hpx::future` to the new and fully conforming version by removing the old code and by renaming the type `hpx::unique_future` to `hpx::future`. In order to maintain backwards compatibility with existing code which uses the type `hpx::unique_future` we support the configuration variable `HPX_UNIQUE_FUTURE_ALIAS`. If this variable is set to `ON` while running cmake it will additionally define a template alias for this type.

• We rewrote and significantly changed our build system. Please have a look at the new (now generated) documentation here: HPX build system. Please revisit your build scripts to adapt to the changes. The most notable changes are:
  
  – `HPX_NO_INSTALL` is no longer necessary.
  
  – For external builds, you need to set `HPX_DIR` instead of `HPX_ROOT` as described here: Using HPX with CMake-based projects.
  
  – IDEs that support multiple configurations (Visual Studio and XCode) can now be used as intended. that means no build dir.
  
  – Building HPX statically (without dynamic libraries) is now supported (`-DHPX_STATIC_LINKING=On`).
  
  – Please note that many variables used to configure the build process have been renamed to unify the naming conventions (see the section CMake variables used to configure HPX for more information).
  
  – This also fixes a long list of issues, for more information see Issue #1204

• We started to implement various proposals to the C++ Standardization committee related to parallelism and concurrency, most notably N4409 (Working Draft, Technical Specification for C++ Extensions for Parallelism), N4411 (Task Region Rev. 3), and N4313 (Working Draft, Technical Specification for C++ Extensions for Concurrency).

• We completely remodeled our automatic build system to run builds and unit tests on various systems and compilers. This allows us to find most bugs right as they were introduced and helps to maintain a high level of quality and compatibility. The newest build logs can be found at HPX Buildbot Website.

Bug fixes (closed tickets)

Here is a list of the important tickets we closed for this release.

• Issue #1296 - Rename `make_error_future` to `make_exceptional_future`, adjust to N4123

• Issue #1295 - building issue

• Issue #1293 - Transpose example

• Issue #1292 - Wrong `abs()` function used in example

3075 https://github.com/STEllAR-GROUP/hpx/issues/1204
3076 http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2015/n4409.pdf
3078 http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2014/n4313.html
3079 http://rostam.cct.lsu.edu/
3080 https://github.com/STEllAR-GROUP/hpx/issues/1296
3081 https://github.com/STEllAR-GROUP/hpx/issues/1295
3082 https://github.com/STEllAR-GROUP/hpx/issues/1293
3083 https://github.com/STEllAR-GROUP/hpx/issues/1292
• Issue #1291 - non-synchronized shift operators have been removed
• Issue #1290 - RDTSCP is defined as true for Xeon Phi build
• Issue #1289 - Fixing 1288
• Issue #1288 - Add new performance counters
• Issue #1287 - Hierarchy scheduler broken performance counters
• Issue #1286 - Algorithm cleanup
• Issue #1285 - Broken Links in Documentation
• Issue #1284 - Uninitialized copy
• Issue #1283 - missing boost::scoped_ptr includes
• Issue #1282 - Update documentation of build options for schedulers
• Issue #1281 - reset idle rate counter
• Issue #1280 - Bug when executing on Intel MIC
• Issue #1279 - Add improved when_all/wait_all
• Issue #1278 - Implement improved when_all/wait_all
• Issue #1277 - feature request: get access to argc argv and variables_map
• Issue #1276 - Remove merging map
• Issue #1274 - Weird (wrong) string code in papi.cpp
• Issue #1273 - Sequential task execution policy
• Issue #1272 - Avoid CMake name clash for Boost.Thread library
• Issue #1271 - Updates on HPX Test Units
• Issue #1270 - hpx/util/safe_lexical_cast.hpp is added
• Issue #1269 - Added default value for “LIB” cmake variable
• Issue #1268 - Memory Counters not working

https://github.com/STEllAR-GROUP/hpx/issues/1291
https://github.com/STEllAR-GROUP/hpx/issues/1290
https://github.com/STEllAR-GROUP/hpx/issues/1289
https://github.com/STEllAR-GROUP/hpx/issues/1288
https://github.com/STEllAR-GROUP/hpx/issues/1287
https://github.com/STEllAR-GROUP/hpx/issues/1286
https://github.com/STEllAR-GROUP/hpx/issues/1285
https://github.com/STEllAR-GROUP/hpx/issues/1284
https://github.com/STEllAR-GROUP/hpx/issues/1283
https://github.com/STEllAR-GROUP/hpx/issues/1282
https://github.com/STEllAR-GROUP/hpx/issues/1281
https://github.com/STEllAR-GROUP/hpx/issues/1280
https://github.com/STEllAR-GROUP/hpx/issues/1279
https://github.com/STEllAR-GROUP/hpx/issues/1278
https://github.com/STEllAR-GROUP/hpx/issues/1277
https://github.com/STEllAR-GROUP/hpx/issues/1276
https://github.com/STEllAR-GROUP/hpx/issues/1274
https://github.com/STEllAR-GROUP/hpx/issues/1273
https://github.com/STEllAR-GROUP/hpx/issues/1272
https://github.com/STEllAR-GROUP/hpx/issues/1271
https://github.com/STEllAR-GROUP/hpx/issues/1270
https://github.com/STEllAR-GROUP/hpx/issues/1269
https://github.com/STEllAR-GROUP/hpx/issues/1268
• Issue #1266 - FindHPX.cmake is not installed
• Issue #1263 - apply_remote test takes too long
• Issue #1262 - Chrono cleanup
• Issue #1261 - Need make install for papi counters and this builds all the examples
• Issue #1260 - Documentation of Stencil example claims
• Issue #1259 - Avoid double-linking Boost on Windows
• Issue #1257 - Adding additional parameter to create_thread
• Issue #1256 - added buildbot changes to release notes
• Issue #1255 - Cannot build MiniGhost
• Issue #1253 - hpx::thread defects
• Issue #1252 - HPX_PREFIX is too fragile
• Issue #1250 - switch_to_fiber_emulation does not work properly
• Issue #1249 - Documentation is generated under Release folder
• Issue #1248 - Fix usage of hpx_generic_coroutine_context and get tests passing on powerpc
• Issue #1247 - Dynamic linking error
• Issue #1246 - Make cpuid.cpp C++11 compliant
• Issue #1245 - HPX fails on startup (setting thread affinity mask)
• Issue #1244 - HPX_WITH_RDTSC configure test fails, but should succeed
• Issue #1243 - CTest dashboard info for CSCS CDash drop location
• Issue #1242 - Mac fixes
• Issue #1241 - Failure in Distributed with Boost 1.56
• Issue #1240 - fix a race condition in examples.diskperf
• Issue #1239 - fix wait_each in examples.diskperf

https://github.com/STEllAR-GROUP/hpx/issues/1266
https://github.com/STEllAR-GROUP/hpx/issues/1263
https://github.com/STEllAR-GROUP/hpx/issues/1262
https://github.com/STEllAR-GROUP/hpx/issues/1261
https://github.com/STEllAR-GROUP/hpx/issues/1260
https://github.com/STEllAR-GROUP/hpx/issues/1259
https://github.com/STEllAR-GROUP/hpx/issues/1258
https://github.com/STEllAR-GROUP/hpx/issues/1257
https://github.com/STEllAR-GROUP/hpx/issues/1256
https://github.com/STEllAR-GROUP/hpx/issues/1255
https://github.com/STEllAR-GROUP/hpx/issues/1254
https://github.com/STEllAR-GROUP/hpx/issues/1253
https://github.com/STEllAR-GROUP/hpx/issues/1252
https://github.com/STEllAR-GROUP/hpx/issues/1251
https://github.com/STEllAR-GROUP/hpx/issues/1250
https://github.com/STEllAR-GROUP/hpx/issues/1249
https://github.com/STEllAR-GROUP/hpx/issues/1248
https://github.com/STEllAR-GROUP/hpx/issues/1247
https://github.com/STEllAR-GROUP/hpx/issues/1246
https://github.com/STEllAR-GROUP/hpx/issues/1245
https://github.com/STEllAR-GROUP/hpx/issues/1244
https://github.com/STEllAR-GROUP/hpx/issues/1243
https://github.com/STEllAR-GROUP/hpx/issues/1242
https://github.com/STEllAR-GROUP/hpx/issues/1241
https://github.com/STEllAR-GROUP/hpx/issues/1240
https://github.com/STEllAR-GROUP/hpx/issues/1239

2.11. Releases
• Issue #1238 - Fixed #1237: hpx::util::portable_binary_iarchive failed
• Issue #1237 - hpx::util::portable_binary_iarchive failed
• Issue #1235 - Fixing clang warnings and errors
• Issue #1234 - TCP runs fail: Transport endpoint is not connected
• Issue #1233 - Making sure the correct number of threads is registered with AGAS
• Issue #1232 - Fixing race in wait_xxx
• Issue #1231 - Parallel minmax
• Issue #1230 - Distributed run of 1d_stencil_8 uses less threads than spec. & sometimes gives errors
• Issue #1229 - Unstable number of threads
• Issue #1228 - HPX link error (cmake / MPI)
• Issue #1226 - Warning about struct/class thread_counters
• Issue #1225 - Adding parallel::replace etc
• Issue #1224 - Extending dataflow to pass through non-future arguments
• Issue #1223 - Remaining find algorithms implemented, N4071
• Issue #1222 - Merging all the changes
• Issue #1221 - No error output when using mpirun with hpx
• Issue #1219 - Adding new AGAS cache performance counters
• Issue #1216 - Fixing using futures (clients) as arguments to actions
• Issue #1215 - Error compiling simple component
• Issue #1214 - Stencil docs
• Issue #1213 - Using more than a few dozen MPI processes on SuperMike results in a seg fault before getting to hpx_main
• Issue #1212 - Parallel rotate
• Issue #1211 - Direct actions cause the future’s shared_state to be leaked
• Issue #1210  - Refactored local::promise to be standard conformant
• Issue #1209  - Improve command line handling
• Issue #1208  - Adding parallel::reverse and parallel::reverse_copy
• Issue #1207  - Add copy_backward and move_backward
• Issue #1206  - N4071 additional algorithms implemented
• Issue #1204  - Cmake simplification and various other minor changes
• Issue #1203  - Implementing new launch policy for (local) async: hpx::launch::fork.
• Issue #1202  - Failed assertion in connection_cache.hpp
• Issue #1201  - pkg-config doesn’t add mpi link directories
• Issue #1200  - Error when querying time performance counters
• Issue #1199  - library path is now configurable (again)
• Issue #1198  - Error when querying performance counters
• Issue #1197  - tests fail with intel compiler
• Issue #1196  - Silence several warnings
• Issue #1195  - Rephrase initializers to work with VC++ 2012
• Issue #1194  - Simplify parallel algorithms
• Issue #1193  - Adding parallel::equal
• Issue #1192  - HPX(out_of_memory) on including <hpx/hpx.hpp>
• Issue #1191  - Fixing #1189
• Issue #1190  - Chrono cleanup
• Issue #1189  - Deadlock .. somewhere? (probably serialization)
• Issue #1188  - Removed future::get_status()
• Issue #1186  - Fixed FindOpenCL to find current AMD APP SDK

https://github.com/STEllAR-GROUP/hpx/issues/1210
https://github.com/STEllAR-GROUP/hpx/issues/1209
https://github.com/STEllAR-GROUP/hpx/issues/1208
https://github.com/STEllAR-GROUP/hpx/issues/1207
https://github.com/STEllAR-GROUP/hpx/issues/1206
https://github.com/STEllAR-GROUP/hpx/issues/1204
https://github.com/STEllAR-GROUP/hpx/issues/1203
https://github.com/STEllAR-GROUP/hpx/issues/1202
https://github.com/STEllAR-GROUP/hpx/issues/1201
https://github.com/STEllAR-GROUP/hpx/issues/1200
https://github.com/STEllAR-GROUP/hpx/issues/1199
https://github.com/STEllAR-GROUP/hpx/issues/1198
https://github.com/STEllAR-GROUP/hpx/issues/1197
https://github.com/STEllAR-GROUP/hpx/issues/1196
https://github.com/STEllAR-GROUP/hpx/issues/1195
https://github.com/STEllAR-GROUP/hpx/issues/1194
https://github.com/STEllAR-GROUP/hpx/issues/1193
https://github.com/STEllAR-GROUP/hpx/issues/1192
https://github.com/STEllAR-GROUP/hpx/issues/1191
https://github.com/STEllAR-GROUP/hpx/issues/1190
https://github.com/STEllAR-GROUP/hpx/issues/1189
https://github.com/STEllAR-GROUP/hpx/issues/1188
https://github.com/STEllAR-GROUP/hpx/issues/1186
• Issue #1184 - Tweaking future unwrapping
• Issue #1183 - Extended `parallel::reduce`
• Issue #1182 - `future::unwrap` hangs for `launch::deferred`
• Issue #1181 - Adding `all_of`, `any_of`, and `none_of` and corresponding documentation
• Issue #1180 - `hpx::cout` defect
• Issue #1179 - `hpx::async` does not work for member function pointers when called on types with self-defined unary `operator*`
• Issue #1178 - Implemented variadic `hpx::util::zip_iterator`
• Issue #1177 - MPI parcelport defect
• Issue #1176 - `HPX_DEFINE_COMPONENT_CONST_ACTION_TPL` does not have a 2-argument version
• Issue #1175 - Create `util::zip_iterator` working with `util::tuple<>`
• Issue #1174 - Error Building HPX on linux, `root_certificate_authority.cpp`
• Issue #1173 - `hpx::cout` output lost
• Issue #1172 - HPX build error with Clang 3.4.2
• Issue #1171 - CMAKE_INSTALL_PREFIX ignored
• Issue #1170 - Close `hpx_benchmarks` repository on Github
• Issue #1169 - Buildbot emails have syntax error in url
• Issue #1167 - Merge partial implementation of standards proposal N3960
• Issue #1166 - Fixed several compiler warnings
• Issue #1165 - `cmake` warns: “tests.regressions.actions” does not exist
• Issue #1164 - Want my own serialization of `hpx::future`
• Issue #1162 - Segfault in `hello_world` example
• Issue #1161 - Use `HPX_ASSERT` to aid the compiler
• Issue #1160 - Do not put -DNDEBUG into `hpx_application.pc`

3176 https://github.com/STEllAR-GROUP/hpx/issues/1184
3177 https://github.com/STEllAR-GROUP/hpx/issues/1183
3178 https://github.com/STEllAR-GROUP/hpx/issues/1182
3179 https://github.com/STEllAR-GROUP/hpx/issues/1181
3180 https://github.com/STEllAR-GROUP/hpx/issues/1180
3181 https://github.com/STEllAR-GROUP/hpx/issues/1179
3182 https://github.com/STEllAR-GROUP/hpx/issues/1178
3183 https://github.com/STEllAR-GROUP/hpx/issues/1177
3184 https://github.com/STEllAR-GROUP/hpx/issues/1176
3185 https://github.com/STEllAR-GROUP/hpx/issues/1175
3186 https://github.com/STEllAR-GROUP/hpx/issues/1174
3187 https://github.com/STEllAR-GROUP/hpx/issues/1173
3188 https://github.com/STEllAR-GROUP/hpx/issues/1172
3189 https://github.com/STEllAR-GROUP/hpx/issues/1171
3190 https://github.com/STEllAR-GROUP/hpx/issues/1170
3191 https://github.com/STEllAR-GROUP/hpx/issues/1169
3192 https://github.com/STEllAR-GROUP/hpx/issues/1168
3193 https://github.com/STEllAR-GROUP/hpx/issues/1167
3194 https://github.com/STEllAR-GROUP/hpx/issues/1166
3195 https://github.com/STEllAR-GROUP/hpx/issues/1165
3196 https://github.com/STEllAR-GROUP/hpx/issues/1164
3197 https://github.com/STEllAR-GROUP/hpx/issues/1162
3198 https://github.com/STEllAR-GROUP/hpx/issues/1161
• Issue #1159 - Support Clang 3.4.2
• Issue #1158 - Fixed #1157: Rename when_n/wait_n, add when_xxx_n/wait_xxx_n
• Issue #1157 - Rename when_n/wait_n, add when_xxx_n/wait_xxx_n
• Issue #1156 - Force inlining fails
• Issue #1155 - changed header of printout to be compatible with python csv module
• Issue #1154 - Fixing iostreams
• Issue #1153 - Standard manipulators (like std::endl) do not work with hpx::ostream
• Issue #1152 - Functions revamp
• Issue #1151 - Supressing cmake 3.0 policy warning for CMP0026
• Issue #1150 - Client Serialization error
• Issue #1149 - Segfault on Stampede
• Issue #1148 - Refactoring mini-ghost
• Issue #1147 - N3960 copy_if and copy_n implemented and tested
• Issue #1146 - Stencil print
• Issue #1145 - N3960 hpx::parallel::copy implemented and tested
• Issue #1144 - OpenMP examples 1d_stencil do not build
• Issue #1143 - 1d_stencil OpenMP examples do not build
• Issue #1142 - Cannot build HPX with gcc 4.6 on OS X
• Issue #1140 - Fix OpenMP lookup, enable usage of config tests in external CMake projects.
• Issue #1139 - hpx/hpx/config/compiler_specific.hpp
• Issue #1138 - clean up pkg-config files
• Issue #1137 - Improvements to create binary packages
• Issue #1136 - HPX_GCC_VERSION not defined on all compilers

https://github.com/STEllAR-GROUP/hpx/issues/1159
https://github.com/STEllAR-GROUP/hpx/issues/1158
https://github.com/STEllAR-GROUP/hpx/issues/1157
https://github.com/STEllAR-GROUP/hpx/issues/1156
https://github.com/STEllAR-GROUP/hpx/issues/1155
https://github.com/STEllAR-GROUP/hpx/issues/1154
https://github.com/STEllAR-GROUP/hpx/issues/1153
https://github.com/STEllAR-GROUP/hpx/issues/1152
https://github.com/STEllAR-GROUP/hpx/issues/1151
https://github.com/STEllAR-GROUP/hpx/issues/1150
https://github.com/STEllAR-GROUP/hpx/issues/1149
https://github.com/STEllAR-GROUP/hpx/issues/1148
https://github.com/STEllAR-GROUP/hpx/issues/1147
https://github.com/STEllAR-GROUP/hpx/issues/1146
https://github.com/STEllAR-GROUP/hpx/issues/1145
https://github.com/STEllAR-GROUP/hpx/issues/1144
https://github.com/STEllAR-GROUP/hpx/issues/1143
https://github.com/STEllAR-GROUP/hpx/issues/1142
https://github.com/STEllAR-GROUP/hpx/issues/1140
https://github.com/STEllAR-GROUP/hpx/issues/1139
https://github.com/STEllAR-GROUP/hpx/issues/1138
https://github.com/STEllAR-GROUP/hpx/issues/1137
https://github.com/STEllAR-GROUP/hpx/issues/1136
- Issue #1135 - Avoiding collision between winsock2.h and windows.h
- Issue #1134 - Making sure, that hpx::finalize can be called from any locality
- Issue #1133 - 1d stencil examples
- Issue #1131 - Refactor unique_function implementation
- Issue #1130 - Unique function
- Issue #1129 - Some fixes to the Build system on OS X
- Issue #1128 - Action future args
- Issue #1127 - Executor causes segmentation fault
- Issue #1124 - Adding new API functions: register_id_with_basename, unregister_id_with_basename, find_ids_from_basename; adding test
- Issue #1123 - Reduce nesting of try-catch construct in encode_parcel?
- Issue #1122 - Client base fixes
- Issue #1121 - Update hpxrun.py.in
- Issue #1120 - HTTS2 tests compile errors on v110 (VS2012)
- Issue #1119 - Remove references to boost::atomic in accumulator example
- Issue #1118 - Only build test thread_pool_executor_1114_test if HPX_LOCAL_SCHEDULER is set
- Issue #1117 - local_queue_executor linker error on vc110
- Issue #1116 - Disabled performance counter should give runtime errors, not invalid data
- Issue #1115 - Compile error with Intel C++ 13.1
- Issue #1114 - Default constructed executor is not usable
- Issue #1113 - Fast compilation of logging causes ABI incompatibilities between different NDEBUG values
- Issue #1112 - Using thread_pool_executors causes segfault
- Issue #1111 - hpx::threads::get_thread_data always returns zero
- Issue #1110 - Remove unnecessary null pointer checks
• Issue #1109 - More tests adjustments
• Issue #1108 - Clarify build rules for “libboost_atomic-mt.so”?
• Issue #1107 - Remove unnecessary null pointer checks
• Issue #1106 - network_storage benchmark improvements, adding legends to plots and tidying layout
• Issue #1105 - Add more plot outputs and improve instructions doc
• Issue #1104 - Complete quoting for parameters of some CMake commands
• Issue #1103 - Work on test/scripts
• Issue #1102 - Changed minimum requirement of window install to 2012
• Issue #1101 - Changed minimum requirement of window install to 2012
• Issue #1100 - Changed readme to no longer specify using MSVC 2010 compiler
• Issue #1099 - Error returning futures from component actions
• Issue #1098 - Improve storage test
• Issue #1097 - data_actions quickstart example calls missing function decorate_action of data_get_action
• Issue #1096 - MPI parcelport broken with new zero copy optimization
• Issue #1095 - Warning C4005: _WIN32_WINNT: Macro redefinition
• Issue #1094 - Syntax error for -DHPX_UNIQUE_FUTURE_ALIAS in master
• Issue #1093 - Syntax error for -DHPX_UNIQUE_FUTURE_ALIAS
• Issue #1092 - Rename unique_future<> back to future<>
• Issue #1091 - Inconsistent error message
• Issue #1090 - On windows 8.1 the examples crashed if using more than one os thread
• Issue #1089 - Components should be allowed to have their own executor
• Issue #1088 - Add possibility to select a network interface for the ibverbs parcelport
• Issue #1087 - ibverbs and ipc parcelport uses zero copy optimization

https://github.com/STEllAR-GROUP/hpx/issues/1109
https://github.com/STEllAR-GROUP/hpx/issues/1108
https://github.com/STEllAR-GROUP/hpx/issues/1107
https://github.com/STEllAR-GROUP/hpx/issues/1106
https://github.com/STEllAR-GROUP/hpx/issues/1105
https://github.com/STEllAR-GROUP/hpx/issues/1104
https://github.com/STEllAR-GROUP/hpx/issues/1103
https://github.com/STEllAR-GROUP/hpx/issues/1102
https://github.com/STEllAR-GROUP/hpx/issues/1101
https://github.com/STEllAR-GROUP/hpx/issues/1100
https://github.com/STEllAR-GROUP/hpx/issues/1099
https://github.com/STEllAR-GROUP/hpx/issues/1098
https://github.com/STEllAR-GROUP/hpx/issues/1097
https://github.com/STEllAR-GROUP/hpx/issues/1096
https://github.com/STEllAR-GROUP/hpx/issues/1095
https://github.com/STEllAR-GROUP/hpx/issues/1094
https://github.com/STEllAR-GROUP/hpx/issues/1093
https://github.com/STEllAR-GROUP/hpx/issues/1092
https://github.com/STEllAR-GROUP/hpx/issues/1091
https://github.com/STEllAR-GROUP/hpx/issues/1090
https://github.com/STEllAR-GROUP/hpx/issues/1089
https://github.com/STEllAR-GROUP/hpx/issues/1088
https://github.com/STEllAR-GROUP/hpx/issues/1087

2.11. Releases
• Issue #1083 - Make shell examples copyable in docs
• Issue #1082 - Implement proper termination detection during shutdown
• Issue #1081 - Implement thread_specific_ptr for hpx::threads
• Issue #1072 - make install not working properly
• Issue #1070 - Complete quoting for parameters of some CMake commands
• Issue #1059 - Fix more unused variable warnings
• Issue #1051 - Implement when_each
• Issue #973 - Would like option to report hwloc bindings
• Issue #970 - Bad flags for Fortran compiler
• Issue #941 - Create a proper user level context switching class for BG/Q
• Issue #935 - Build error with gcc 4.6 and Boost 1.54.0 on hpx trunk and 0.9.6
• Issue #934 - Want to build HPX without dynamic libraries
• Issue #927 - Make hpx/lcos/reduce.hpp accept futures of id_type
• Issue #926 - All unit tests that are run with more than one thread with CTest/hpx_run_test should configure hpx.os_threads
• Issue #925 - regression_dataflow_791 needs to be brought in line with HPX standards
• Issue #899 - Fix race conditions in regression tests
• Issue #879 - Hung test leads to cascading test failure; make tests should support the MPI parcelport
• Issue #865 - future<T> and friends shall work for movable only Ts
• Issue #847 - Dynamic libraries are not installed on OS X
• Issue #816 - First Program tutorial pull request
• Issue #799 - Wrap lexical_cast to avoid exceptions
• Issue #720 - broken configuration when using ccmake on Ubuntu
• Issue #622 - --hpx:hpx and --hpx:debug-hpx-log is nonsensical
• Issue #5253291 - Extend barrier LCO test to run in distributed
• Issue #5153292 - Multi-destination version of hpx::apply is broken
• Issue #5093293 - Push Boost.Atomic changes upstream
• Issue #5033294 - Running HPX applications on Windows should not require setting %PATH%
• Issue #4613295 - Add a compilation sanity test
• Issue #4563296 - hpx_run_tests.py should log output from tests that timeout
• Issue #4543297 - Investigate threadmanager performance
• Issue #3453298 - Add more versatile environmental/cmake variable support to hpx_find_* CMake macros
• Issue #2093299 - Support multiple configurations in generated build files
• Issue #1903300 - hpx::cout should be a std::ostream
• Issue #1893301 - iostreams component should use startup/shutdown functions
• Issue #1833302 - Use Boost.ICL for correctness in AGAS
• Issue #443303 - Implement real futures

2.11.12 HPX V0.9.8 (Mar 24, 2014)

We have had over 800 commits since the last release and we have closed over 65 tickets (bugs, feature requests, etc.). With the changes below, HPX is once again leading the charge of a whole new era of computation. By intrinsically breaking down and synchronizing the work to be done, HPX insures that application developers will no longer have to fret about where a segment of code executes. That allows coders to focus their time and energy to understanding the data dependencies of their algorithms and thereby the core obstacles to an efficient code. Here are some of the advantages of using HPX:

• HPX is solidly rooted in a sophisticated theoretical execution model – ParalleX

• HPX exposes an API fully conforming to the C++11 and the draft C++14 standards, extended and applied to distributed computing. Everything programmers know about the concurrency primitives of the standard C++ library is still valid in the context of HPX.

• It provides a competitive, high performance implementation of modern, future-proof ideas which gives an smooth migration path from todays mainstream techniques

• There is no need for the programmer to worry about lower level parallelization paradigms like threads or message passing; no need to understand pthreads, MPI, OpenMP, or Windows threads, etc.

• There is no need to think about different types of parallelism such as tasks, pipelines, or fork-join, task or data parallelism.

• The same source of your program compiles and runs on Linux, BlueGene/Q, Mac OS X, Windows, and Android.

3291 https://github.com/STEllAR-GROUP/hpx/issues/525
3292 https://github.com/STEllAR-GROUP/hpx/issues/515
3293 https://github.com/STEllAR-GROUP/hpx/issues/509
3294 https://github.com/STEllAR-GROUP/hpx/issues/503
3295 https://github.com/STEllAR-GROUP/hpx/issues/461
3296 https://github.com/STEllAR-GROUP/hpx/issues/456
3297 https://github.com/STEllAR-GROUP/hpx/issues/454
3298 https://github.com/STEllAR-GROUP/hpx/issues/345
3299 https://github.com/STEllAR-GROUP/hpx/issues/209
3300 https://github.com/STEllAR-GROUP/hpx/issues/190
3301 https://github.com/STEllAR-GROUP/hpx/issues/189
3302 https://github.com/STEllAR-GROUP/hpx/issues/183
3303 https://github.com/STEllAR-GROUP/hpx/issues/44
• The same code runs on shared memory multi-core systems and supercomputers, on handheld devices and Intel® Xeon Phi™ accelerators, or a heterogeneous mix of those.

General changes

• A major API breaking change for this release was introduced by implementing `hpx::future` and `hpx::shared_future` fully in conformance with the C++11 Standard\(^{3304}\). While `hpx::shared_future` is new and will not create any compatibility problems, we revised the interface and implementation of the existing `hpx::future`. For more details please see the mailing list archive\(^{3305}\). To avoid any incompatibilities for existing code we named the type which implements the `std::future` interface as `hpx::unique_future`. For the next release this will be renamed to `hpx::future`, making it full conforming to C++11 Standard\(^{3306}\).

• A large part of the code base of HPX has been refactored and partially re-implemented. The main changes were related to
  – The threading subsystem: these changes significantly reduce the amount of overheads caused by the schedulers, improve the modularity of the code base, and extend the variety of available scheduling algorithms.
  – The parcel subsystem: these changes improve the performance of the HPX networking layer, modularize the structure of the parcelports, and simplify the creation of new parcelports for other underlying networking libraries.
  – The API subsystem: these changes improved the conformance of the API to C++11 Standard, extend and unify the available API functionality, and decrease the overheads created by various elements of the API.
  – The robustness of the component loading subsystem has been improved significantly, allowing to more portably and more reliably register the components needed by an application as startup. This additionally speeds up general application initialization.

• We added new API functionality like `hpx::migrate` and `hpx::copy_component` which are the basic building blocks necessary for implementing higher level abstractions for system-wide load balancing, runtime-adaptive resource management, and object-oriented checkpointing and state-management.

• We removed the use of C++11 move emulation (using Boost.Move), replacing it with C++11 rvalue references. This is the first step towards using more and more native C++11 facilities which we plan to introduce in the future.

• We improved the reference counting scheme used by HPX which helps managing distributed objects and memory. This improves the overall stability of HPX and further simplifies writing real world applications.

• The minimal Boost version required to use HPX is now V1.49.0.

• This release coincides with the first release of HPXPI (V0.1.0), the first implementation of the XPI specification\(^{3307}\).

Bug fixes (closed tickets)

Here is a list of the important tickets we closed for this release.

• Issue #1086\(^{3308}\) - Expose internal boost::shared_array to allow user management of array lifetime
• Issue #1083\(^{3309}\) - Make shell examples copyable in docs
• Issue #1080\(^{3310}\) - /threads{locality#*/total}/count/cumulative broken

---

\(^{3304}\) [http://www.open-std.org/jtc1/sc22/wg21](http://www.open-std.org/jtc1/sc22/wg21)
\(^{3305}\) [http://mail.cct.lsu.edu/pipermail/hpx-users/2014-January/000141.html](http://mail.cct.lsu.edu/pipermail/hpx-users/2014-January/000141.html)
\(^{3306}\) [http://www.open-std.org/jtc1/sc22/wg21](http://www.open-std.org/jtc1/sc22/wg21)
\(^{3308}\) [https://github.com/STEllAR-GROUP/hpx/issues/1086](https://github.com/STEllAR-GROUP/hpx/issues/1086)
\(^{3309}\) [https://github.com/STEllAR-GROUP/hpx/issues/1083](https://github.com/STEllAR-GROUP/hpx/issues/1083)
\(^{3310}\) [https://github.com/STEllAR-GROUP/hpx/issues/1080](https://github.com/STEllAR-GROUP/hpx/issues/1080)
- Issue #1079 - Build problems on OS X
- Issue #1078 - Improve robustness of component loading
- Issue #1077 - Fix a missing enum definition for ‘take’ mode
- Issue #1076 - Merge Jb master
- Issue #1075 - Unknown CMake command “add_hpx_pseudo_target”
- Issue #1074 - Implement `apply_continue_callback` and `apply_colocated_callback`
- Issue #1073 - The new `apply_colocated` and `async_colocated` functions lead to automatic registered functions
- Issue #1071 - Remove deferred_packaged_task
- Issue #1069 - serialize_buffer with allocator fails at destruction
- Issue #1068 - Coroutine include and forward declarations missing
- Issue #1067 - Add allocator support to `util::serialize_buffer`
- Issue #1066 - Allow for MPI_Init being called before HPX launches
- Issue #1065 - AGAS cache isn’t used/populated on worker localities
- Issue #1064 - Reorder includes to ensure ws2 includes early
- Issue #1063 - Add `hpx::runtime::suspend` and `hpx::runtime::resume`
- Issue #1062 - Fix `async_continue` to properly handle return types
- Issue #1061 - Implement `async_colocated` and `apply_colocated`
- Issue #1060 - Implement minimal component migration
- Issue #1059 - Remove `HPX_UTIL_TUPLE` from code base
- Issue #1057 - Add performance counters for threading subsystem
- Issue #1055 - Thread allocation uses two memory pools
- Issue #1053 - Work stealing flawed
- Issue #1052 - Fix a number of warnings

3311 https://github.com/STEllAR-GROUP/hpx/issues/1079
3312 https://github.com/STEllAR-GROUP/hpx/issues/1078
3313 https://github.com/STEllAR-GROUP/hpx/issues/1077
3314 https://github.com/STEllAR-GROUP/hpx/issues/1076
3315 https://github.com/STEllAR-GROUP/hpx/issues/1075
3316 https://github.com/STEllAR-GROUP/hpx/issues/1074
3317 https://github.com/STEllAR-GROUP/hpx/issues/1073
3318 https://github.com/STEllAR-GROUP/hpx/issues/1071
3319 https://github.com/STEllAR-GROUP/hpx/issues/1069
3320 https://github.com/STEllAR-GROUP/hpx/issues/1068
3321 https://github.com/STEllAR-GROUP/hpx/issues/1067
3322 https://github.com/STEllAR-GROUP/hpx/issues/1066
3323 https://github.com/STEllAR-GROUP/hpx/issues/1065
3324 https://github.com/STEllAR-GROUP/hpx/issues/1064
3325 https://github.com/STEllAR-GROUP/hpx/issues/1063
3326 https://github.com/STEllAR-GROUP/hpx/issues/1062
3327 https://github.com/STEllAR-GROUP/hpx/issues/1061
3328 https://github.com/STEllAR-GROUP/hpx/issues/1060
3329 https://github.com/STEllAR-GROUP/hpx/issues/1059
3330 https://github.com/STEllAR-GROUP/hpx/issues/1057
3331 https://github.com/STEllAR-GROUP/hpx/issues/1055
3332 https://github.com/STEllAR-GROUP/hpx/issues/1053
3333 https://github.com/STEllAR-GROUP/hpx/issues/1052
• Issue #1049 - Fixes for TLS on OSX and more reliable test running
• Issue #1048 - Fixing after 588 hang
• Issue #1047 - Use port ‘0’ for networking when using one locality
• Issue #1046 - `composable_guard` test is broken when having more than one thread
• Issue #1045 - Security missing headers
• Issue #1044 - Native TLS on FreeBSD via __thread
• Issue #1043 - `async` et.al. compute the wrong result type
• Issue #1042 - `async` et.al. implicitly unwrap reference wrappers
• Issue #1041 - Remove redundant costly Kleene stars from regex searches
• Issue #1040 - CMakscript regex match patterns has unnecessary kleenes
• Issue #1039 - Remove use of Boost.Move and replace with std::move and real rvalue refs
• Issue #1038 - Bump minimal required Boost to 1.49.0
• Issue #1037 - Implicit unwrapping of futures in async broken
• Issue #1036 - Scheduler hangs when user code attempts to “block” OS-threads
• Issue #1035 - Idle-rate counter always reports 100% idle rate
• Issue #1034 - Symbolic name registration causes application hangs
• Issue #1033 - Application options read in from an options file generate an error message
• Issue #1032 - `hpx::id_type` local reference counting is wrong
• Issue #1031 - Negative entry in reference count table
• Issue #1030 - Implement condition_variable
• Issue #1029 - Deadlock in thread scheduling subsystem
• Issue #1028 - HPX-thread cumulative count performance counters report incorrect value
• Issue #1027 - `Exposé hpx::thread_interrupted` error code as a separate exception type
• Issue #1026 - Exceptions thrown in asynchronous calls can be lost if the value of the future is never queried
• Issue #1025 - future::wait_for/wait_until do not remove callback
• Issue #1024 - Remove dependence to boost assert and create hpx assert
• Issue #1023 - Segfaults with tcmalloc
• Issue #1022 - prerequisites link in readme is broken
• Issue #1020 - HPX Deadlock on external synchronization
• Issue #1019 - Convert using BOOST_ASSERT to HPX_ASSERT
• Issue #1018 - compiling bug with gcc 4.8.1
• Issue #1017 - Possible crash in io_pool executor
• Issue #1016 - Crash at startup
• Issue #1014 - Implement Increment/Decrement Merging
• Issue #1013 - Add more logging channels to enable greater control over logging granularity
• Issue #1012 - --hpx:debug-hpx-log and --hpx:debug-agas-log lead to non-thread safe writes
• Issue #1011 - After installation, running applications from the build/staging directory no longer works
• Issue #1010 - Mergable decrement requests are not being merged
• Issue #1009 - --hpx:list-symbolic-names crashes
• Issue #1007 - Components are not properly destroyed
• Issue #1006 - Segfault/hang in set_data
• Issue #1003 - Performance counter naming issue
• Issue #982 - Race condition during startup
• Issue #912 - OS X: component type not found in map
• Issue #663 - Create a buildbot slave based on Clang 3.2/OSX
• Issue #636 - Expose this_locality::apply<act>(p1, p2); for local execution
• Issue #197\(3380\) - Add \(--\text{console}=	ext{address}\) option for PBS runs
• Issue #175\(3381\) - Asynchronous AGAS API

**2.11.13 HPX V0.9.7 (Nov 13, 2013)**

We have had over 1000 commits since the last release and we have closed over 180 tickets (bugs, feature requests, etc.).

**General changes**

• Ported HPX to BlueGene/Q
• Improved HPX support for Xeon/Phi accelerators
  • Reimplemented `hpx::bind`, `hpx::tuple`, and `hpx::function` for better performance and better compliance with the C++11 Standard. Added `hpx::mem_fn`.
  • Reworked `hpx::when_all` and `hpx::when_any` for better compliance with the ongoing C++ standardization effort, added heterogeneous version for those functions. Added `hpx::when_any_swapped`.
• Added `hpx::copy` as a precursor for a migrate functionality
  • Added `hpx::get_ptr` allowing to directly access the memory underlying a given component
  • Added the `hpx::lcos::broadcast`, `hpx::lcos::reduce`, and `hpx::lcos::fold` collective operations
  • Added `hpx::get_locality_name` allowing to retrieve the name of any of the localities for the application.
• Added support for more flexible thread affinity control from the HPX command line, such as new modes for \(--\text{hpx:bind} \) (balanced, scattered, compact), improved default settings when running multiple localities on the same node.
• Added experimental executors for simpler thread pooling and scheduling. This API may change in the future as it will stay aligned with the ongoing C++ standardization efforts.
• Massively improved the performance of the HPX serialization code. Added partial support for zero copy serialization of array and bitwise-copyable types.
• General performance improvements of the code related to threads and futures.

**Bug fixes (closed tickets)**

Here is a list of the important tickets we closed for this release.

• Issue #1005\(3382\) - Allow to disable array optimizations and zero copy optimizations for each parcelport
• Issue #1004\(3383\) - Generate new HPX logo image for the docs
• Issue #1002\(3384\) - If MPI parcelport is not available, running HPX under mpirun should fail
• Issue #1001\(3385\) - Zero copy serialization raises assert
• Issue #1000\(3386\) - Can’t connect to a HPX application running with the MPI parcelport from a non MPI parcelport locality

\(3380\) https://github.com/STEllAR-GROUP/hpx/issues/197
\(3381\) https://github.com/STEllAR-GROUP/hpx/issues/175
\(3382\) https://github.com/STEllAR-GROUP/hpx/issues/1005
\(3383\) https://github.com/STEllAR-GROUP/hpx/issues/1004
\(3384\) https://github.com/STEllAR-GROUP/hpx/issues/1002
\(3385\) https://github.com/STEllAR-GROUP/hpx/issues/1001
\(3386\) https://github.com/STEllAR-GROUP/hpx/issues/1000
• Issue #999\textsuperscript{3387} - Optimize `hpx::when_n`
• Issue #998\textsuperscript{3388} - Fixed const-correctness
• Issue #997\textsuperscript{3389} - Making `serialize_buffer::data()` type save
• Issue #996\textsuperscript{3390} - Memory leak in `hpx::lcos::promise`
• Issue #995\textsuperscript{3391} - Race while registering pre-shutdown functions
• Issue #994\textsuperscript{3392} - thread_rescheduling regression test does not compile
• Issue #992\textsuperscript{3393} - Correct comments and messages
• Issue #991\textsuperscript{3394} - `setcap cap_sys_rawio=ep` for power profiling causes an HPX application to abort
• Issue #989\textsuperscript{3395} - Jacobi hangs during execution
• Issue #988\textsuperscript{3396} - `multiple_init` test is failing
• Issue #986\textsuperscript{3397} - Can’t call a function called “init” from “main” when using `<hpx/hpx_main.hpp>`
• Issue #984\textsuperscript{3398} - Reference counting tests are failing
• Issue #983\textsuperscript{3399} - `thread_suspension_executor` test fails
• Issue #980\textsuperscript{3400} - Terminating HPX threads don’t leave stack in virgin state
• Issue #979\textsuperscript{3401} - Static scheduler not in documents
• Issue #978\textsuperscript{3402} - Preprocessing limits are broken
• Issue #977\textsuperscript{3403} - Make `tests.regressions.lcos.future_hang_on_get` shorter
• Issue #976\textsuperscript{3404} - Wrong library order in `pkgconfig`
• Issue #975\textsuperscript{3405} - Please reopen #963
• Issue #974\textsuperscript{3406} - Option `pu-offset` ignored in `fixing_588` branch
• Issue #972\textsuperscript{3407} - Cannot use MKL with HPX
• Issue #969\textsuperscript{3408} - Non-existent INI files requested on the command line via `--hpx:config` do not cause warnings or errors.
• Issue #968\textsuperscript{3409} - Cannot build examples in `fixing_588` branch

\textsuperscript{3387}https://github.com/STEllAR-GROUP/hpx/issues/999
\textsuperscript{3388}https://github.com/STEllAR-GROUP/hpx/issues/998
\textsuperscript{3389}https://github.com/STEllAR-GROUP/hpx/issues/997
\textsuperscript{3390}https://github.com/STEllAR-GROUP/hpx/issues/996
\textsuperscript{3391}https://github.com/STEllAR-GROUP/hpx/issues/995
\textsuperscript{3392}https://github.com/STEllAR-GROUP/hpx/issues/994
\textsuperscript{3393}https://github.com/STEllAR-GROUP/hpx/issues/993
\textsuperscript{3394}https://github.com/STEllAR-GROUP/hpx/issues/992
\textsuperscript{3395}https://github.com/STEllAR-GROUP/hpx/issues/991
\textsuperscript{3396}https://github.com/STEllAR-GROUP/hpx/issues/989
\textsuperscript{3397}https://github.com/STEllAR-GROUP/hpx/issues/988
\textsuperscript{3398}https://github.com/STEllAR-GROUP/hpx/issues/986
\textsuperscript{3399}https://github.com/STEllAR-GROUP/hpx/issues/984
\textsuperscript{3400}https://github.com/STEllAR-GROUP/hpx/issues/983
\textsuperscript{3401}https://github.com/STEllAR-GROUP/hpx/issues/980
\textsuperscript{3402}https://github.com/STEllAR-GROUP/hpx/issues/979
\textsuperscript{3403}https://github.com/STEllAR-GROUP/hpx/issues/978
\textsuperscript{3404}https://github.com/STEllAR-GROUP/hpx/issues/977
\textsuperscript{3405}https://github.com/STEllAR-GROUP/hpx/issues/976
\textsuperscript{3406}https://github.com/STEllAR-GROUP/hpx/issues/975
\textsuperscript{3407}https://github.com/STEllAR-GROUP/hpx/issues/974
\textsuperscript{3408}https://github.com/STEllAR-GROUP/hpx/issues/969
\textsuperscript{3409}https://github.com/STEllAR-GROUP/hpx/issues/968
• Issue #967 - Command line description of --hpx:queuing seems wrong
• Issue #966 - --hpx:print-bind physical core numbers are wrong
• Issue #965 - Deadlock when building in Release mode
• Issue #963 - Not all worker threads are working
• Issue #962 - Problem with SLURM integration
• Issue #961 - --hpx:print-bind outputs incorrect information
• Issue #960 - Fix cut and paste error in documentation of get_thread_priority
• Issue #959 - Change link to boost.atomic in documentation to point to boost.org
• Issue #958 - Undefined reference to intrusive_ptr_release
• Issue #957 - Make tuple standard compliant
• Issue #956 - Segfault with a3382fb
• Issue #955 - --hpx:node and --hpx:nodetfiles do not work with foreign nodes
• Issue #954 - Make order of arguments for hpx::async and hpx::broadcast consistent
• Issue #953 - Cannot use MKL with HPX
• Issue #952 - register_ [pre_] shutdown_function never throw
• Issue #951 - Assert when number of threads is greater than hardware concurrency
• Issue #948 - HPX_HAVE_GENERIC_CONTEXT_COROUTINES conflicts with HPX_HAVE_FIBER_BASED_COROUTINES
• Issue #947 - Need MPI_THREAD_MULTIPLE for backward compatibility
• Issue #946 - HPX does not call MPI_Finalize
• Issue #945 - Segfault with hpx::lcos::broadcast
• Issue #944 - OS X: assertion pu_offset_ < hardware_concurrency failed
• Issue #943 - #include <hpx/hpx_main.hpp> does not work
• Issue #942 - Make the BG/Q work with -O3
• Issue #940 - Use separator when concatenating locality name
• Issue #939 - Refactor MPI parcelport to use MPI_Wait instead of multiple MPI_Test calls
• Issue #938 - Want to officially access client_base::gid_
• Issue #937 - client_base::gid_ should be private"
• Issue #936 - Want doxygen-like source code index
• Issue #935 - Build error with gcc 4.6 and Boost 1.54.0 on hpx trunk and 0.9.6
• Issue #933 - Cannot build HPX with Boost 1.54.0
• Issue #932 - Components are destructed too early
• Issue #931 - Make HPX work on BG/Q
• Issue #930 - make git-docs is broken
• Issue #929 - Generating index in docs broken
• Issue #928 - Optimize hpx::util::static_ for C++11 compilers supporting magic statics
• Issue #924 - Make kill_process_tree (in process.py) more robust on Mac OSX
• Issue #923 - Correct BLAS and RNPL cmake tests
• Issue #922 - Cannot link against BLAS
• Issue #921 - Implement hpx::mem_fn
• Issue #920 - Output locality with --hpx:print-bind
• Issue #919 - Correct grammar; simplify boolean expressions
• Issue #918 - Link to hello_world.cpp is broken
• Issue #917 - adapt cmake file to new boostbook version
• Issue #916 - fix problem building documentation with xsltproc >= 1.1.27
• Issue #915 - Add another TBBMalloc library search path
• Issue #914 - Build problem with Intel compiler on Stampede (TACC)
• Issue #913 - fix error messages in fibonacci examples
• Issue #911 - Update OS X build instructions
• Issue #910 - Want like to specify MPI_ROOT instead of compiler wrapper script
• Issue #909 - Warning about void* arithmetic
• Issue #908 - Buildbot for MIC is broken
• Issue #906 - Can’t use --hpx:bind=balanced with multiple MPI processes
• Issue #905 - --hpx:bind documentation should describe full grammar
• Issue #904 - Add hpx::lcos::fold and hpx::lcos::inverse_fold collective operation
• Issue #903 - Add hpx::when_any_swapped()
• Issue #902 - Add hpx::lcos::reduce collective operation
• Issue #901 - Web documentation is not searchable
• Issue #900 - Web documentation for trunk has no index
• Issue #898 - Some tests fail with GCC 4.8.1 and MPI parcel port
• Issue #897 - HWLOC causes failures on Mac
• Issue #896 - pu-offset leads to startup error
• Issue #895 - hpx::get_locality_name not defined
• Issue #894 - Race condition at shutdown
• Issue #893 - --hpx:print-bind switches std::cout to hexadecimal mode
• Issue #892 - hwloc_topology_load can be expensive – don’t call multiple times
• Issue #891 - The documentation for get_locality_name is wrong
• Issue #890 - --hpx:print-bind should not exit
• Issue #889 - --hpx:debug-hpx-log=FILE does not work
• Issue #888 - MPI parcelport does not exit cleanly for --hpx:print-bind

https://github.com/STEllAR-GROUP/hpx/issues/913
https://github.com/STEllAR-GROUP/hpx/issues/911
https://github.com/STEllAR-GROUP/hpx/issues/910
https://github.com/STEllAR-GROUP/hpx/issues/909
https://github.com/STEllAR-GROUP/hpx/issues/908
https://github.com/STEllAR-GROUP/hpx/issues/906
https://github.com/STEllAR-GROUP/hpx/issues/905
https://github.com/STEllAR-GROUP/hpx/issues/904
https://github.com/STEllAR-GROUP/hpx/issues/903
https://github.com/STEllAR-GROUP/hpx/issues/902
https://github.com/STEllAR-GROUP/hpx/issues/901
https://github.com/STEllAR-GROUP/hpx/issues/900
https://github.com/STEllAR-GROUP/hpx/issues/898
https://github.com/STEllAR-GROUP/hpx/issues/897
https://github.com/STEllAR-GROUP/hpx/issues/896
https://github.com/STEllAR-GROUP/hpx/issues/895
https://github.com/STEllAR-GROUP/hpx/issues/894
https://github.com/STEllAR-GROUP/hpx/issues/893
https://github.com/STEllAR-GROUP/hpx/issues/892
https://github.com/STEllAR-GROUP/hpx/issues/891
https://github.com/STEllAR-GROUP/hpx/issues/890
https://github.com/STEllAR-GROUP/hpx/issues/889
https://github.com/STEllAR-GROUP/hpx/issues/888
• Issue #887\textsuperscript{3479} - Choose thread affinities more cleverly
• Issue #886\textsuperscript{3480} - Logging documentation is confusing
• Issue #885\textsuperscript{3481} - Two threads are slower than one
• Issue #884\textsuperscript{3482} - is_callable failing with member pointers in C++11
• Issue #883\textsuperscript{3483} - Need help with is_callable_test
• Issue #882\textsuperscript{3484} - tests.regressions.lcos.future\_hang\_on\_get does not terminate
• Issue #881\textsuperscript{3485} - tests/regressions/block\_matrix/matrix.hh won’t compile with GCC 4.8.1
• Issue #880\textsuperscript{3486} - HPX does not work on OS X
• Issue #878\textsuperscript{3487} - future::unwrap triggers assertion
• Issue #877\textsuperscript{3488} - “make tests” has build errors on Ubuntu 12.10
• Issue #876\textsuperscript{3489} - tcmalloc is used by default, even if it is not present
• Issue #875\textsuperscript{3490} - global\_fixture is defined in a header file
• Issue #874\textsuperscript{3491} - Some tests take very long
• Issue #873\textsuperscript{3492} - Add block-matrix code as regression test
• Issue #872\textsuperscript{3493} - HPX documentation does not say how to run tests with detailed output
• Issue #871\textsuperscript{3494} - All tests fail with “make test”
• Issue #870\textsuperscript{3495} - Please explicitly disable serialization in classes that don’t support it
• Issue #868\textsuperscript{3496} - boost\_any test failing
• Issue #867\textsuperscript{3497} - Reduce the number of copies of hpx::function arguments
• Issue #863\textsuperscript{3498} - Futures should not require a default constructor
• Issue #862\textsuperscript{3499} - value\_or\_error shall not default construct its result
• Issue #861\textsuperscript{3500} - HPXUNUSED macro
• Issue #860\textsuperscript{3501} - Add functionality to copy construct a component

\textsuperscript{3479} https://github.com/STEllAR-GROUP/hpx/issues/887
\textsuperscript{3480} https://github.com/STEllAR-GROUP/hpx/issues/886
\textsuperscript{3481} https://github.com/STEllAR-GROUP/hpx/issues/885
\textsuperscript{3482} https://github.com/STEllAR-GROUP/hpx/issues/884
\textsuperscript{3483} https://github.com/STEllAR-GROUP/hpx/issues/883
\textsuperscript{3484} https://github.com/STEllAR-GROUP/hpx/issues/882
\textsuperscript{3485} https://github.com/STEllAR-GROUP/hpx/issues/881
\textsuperscript{3486} https://github.com/STEllAR-GROUP/hpx/issues/880
\textsuperscript{3487} https://github.com/STEllAR-GROUP/hpx/issues/878
\textsuperscript{3488} https://github.com/STEllAR-GROUP/hpx/issues/877
\textsuperscript{3489} https://github.com/STEllAR-GROUP/hpx/issues/876
\textsuperscript{3490} https://github.com/STEllAR-GROUP/hpx/issues/875
\textsuperscript{3491} https://github.com/STEllAR-GROUP/hpx/issues/874
\textsuperscript{3492} https://github.com/STEllAR-GROUP/hpx/issues/873
\textsuperscript{3493} https://github.com/STEllAR-GROUP/hpx/issues/872
\textsuperscript{3494} https://github.com/STEllAR-GROUP/hpx/issues/871
\textsuperscript{3495} https://github.com/STEllAR-GROUP/hpx/issues/870
\textsuperscript{3496} https://github.com/STEllAR-GROUP/hpx/issues/868
\textsuperscript{3497} https://github.com/STEllAR-GROUP/hpx/issues/867
\textsuperscript{3498} https://github.com/STEllAR-GROUP/hpx/issues/863
\textsuperscript{3499} https://github.com/STEllAR-GROUP/hpx/issues/862
\textsuperscript{3500} https://github.com/STEllAR-GROUP/hpx/issues/861
\textsuperscript{3501} https://github.com/STEllAR-GROUP/hpx/issues/860
• Issue #859: `hpx::endl` should flush
• Issue #858: Create `hpx::get_ptr<>` allowing to access component implementation
• Issue #855: Implement `hpx::INVOKE`
• Issue #854: `hpx/hpx.hpp` does not include `hpx/include/iostreams.hpp`
• Issue #853: Feature request: null future
• Issue #852: Feature request: Locality names
• Issue #851: `hpx::cout` output does not appear on screen
• Issue #849: All tests fail on OS X after installing
• Issue #848: Update OS X build instructions
• Issue #846: Update `hpx_external_example`
• Issue #845: Issues with having both debug and release modules in the same directory
• Issue #844: Create configuration header
• Issue #843: Tests should use CTest
• Issue #842: Remove `buffer_pool` from MPI parcelport
• Issue #841: Add possibility to broadcast an index with `hpx::lcos::broadcast`
• Issue #838: Simplify `util::tuple`
• Issue #837: Adopt boost::tuple tests for `util::tuple`
• Issue #836: Adopt boost::function tests for `util::function`
• Issue #835: Tuple interface missing pieces
• Issue #833: Partially preprocessing files not working
• Issue #832: Native papi counters do not work with wild cards
• Issue #831: Arithmetic counter fails if only one parameter is given
• Issue #830: Convert `hpx::util::function` to use new scheme for serializing its base pointer

3502 https://github.com/STEllAR-GROUP/hpx/issues/859
3503 https://github.com/STEllAR-GROUP/hpx/issues/858
3504 https://github.com/STEllAR-GROUP/hpx/issues/855
3505 https://github.com/STEllAR-GROUP/hpx/issues/854
3506 https://github.com/STEllAR-GROUP/hpx/issues/853
3507 https://github.com/STEllAR-GROUP/hpx/issues/852
3508 https://github.com/STEllAR-GROUP/hpx/issues/851
3509 https://github.com/STEllAR-GROUP/hpx/issues/849
3510 https://github.com/STEllAR-GROUP/hpx/issues/848
3511 https://github.com/STEllAR-GROUP/hpx/issues/846
3512 https://github.com/STEllAR-GROUP/hpx/issues/845
3513 https://github.com/STEllAR-GROUP/hpx/issues/844
3514 https://github.com/STEllAR-GROUP/hpx/issues/843
3515 https://github.com/STEllAR-GROUP/hpx/issues/842
3516 https://github.com/STEllAR-GROUP/hpx/issues/841
3517 https://github.com/STEllAR-GROUP/hpx/issues/838
3518 https://github.com/STEllAR-GROUP/hpx/issues/837
3519 https://github.com/STEllAR-GROUP/hpx/issues/836
3520 https://github.com/STEllAR-GROUP/hpx/issues/835
3521 https://github.com/STEllAR-GROUP/hpx/issues/833
3522 https://github.com/STEllAR-GROUP/hpx/issues/832
3523 https://github.com/STEllAR-GROUP/hpx/issues/831
3524 https://github.com/STEllAR-GROUP/hpx/issues/830

1024 Chapter 2. What’s so special about HPX?
- Issue #829 - Consistently use `decay<T>` instead of `remove_const< remove_reference<T>>`
- Issue #828 - Update future implementation to N3721 and N3722
- Issue #827 - Enable MPI parcelport for bootstrapping whenever application was started using mpirun
- Issue #826 - Support command line option `--hpx:print-bind` even if `--hpx::bind` was not used
- Issue #825 - Memory counters give segfault when attempting to use thread wild cards or numbers only total works
- Issue #824 - Enable lambda functions to be used with `hpx::async/hpx::apply`
- Issue #823 - Using a hashing filter
- Issue #822 - Silence unused variable warning
- Issue #821 - Detect if a function object is callable with given arguments
- Issue #820 - Allow wildcards to be used for performance counter names
- Issue #819 - Make the AGAS symbolic name registry distributed
- Issue #818 - Add future::then() overload taking an executor
- Issue #817 - Fixed typo
- Issue #815 - Create an lco that is performing an efficient broadcast of actions
- Issue #814 - Papi counters cannot specify thread#* to get the counts for all threads
- Issue #813 - Scoped unlock
- Issue #811 - `simple_central_tuplespace_client` run error
- Issue #810 - ostream error when `<<` any objects
- Issue #809 - Optimize parcel serialization
- Issue #808 - HPX applications throw exception when executed from the build directory
- Issue #807 - Create performance counters exposing overall AGAS statistics
- Issue #795 - Create timed `make_ready_future`
- Issue #794 - Create heterogeneous `when_all/when_any/etc.`

https://github.com/STEllAR-GROUP/hpx/issues/829
https://github.com/STEllAR-GROUP/hpx/issues/828
https://github.com/STEllAR-GROUP/hpx/issues/827
https://github.com/STEllAR-GROUP/hpx/issues/826
https://github.com/STEllAR-GROUP/hpx/issues/825
https://github.com/STEllAR-GROUP/hpx/issues/824
https://github.com/STEllAR-GROUP/hpx/issues/823
https://github.com/STEllAR-GROUP/hpx/issues/822
https://github.com/STEllAR-GROUP/hpx/issues/821
https://github.com/STEllAR-GROUP/hpx/issues/820
https://github.com/STEllAR-GROUP/hpx/issues/819
https://github.com/STEllAR-GROUP/hpx/issues/818
https://github.com/STEllAR-GROUP/hpx/issues/817
https://github.com/STEllAR-GROUP/hpx/issues/815
https://github.com/STEllAR-GROUP/hpx/issues/814
https://github.com/STEllAR-GROUP/hpx/issues/813
https://github.com/STEllAR-GROUP/hpx/issues/811
https://github.com/STEllAR-GROUP/hpx/issues/810
https://github.com/STEllAR-GROUP/hpx/issues/809
https://github.com/STEllAR-GROUP/hpx/issues/808
https://github.com/STEllAR-GROUP/hpx/issues/795
https://github.com/STEllAR-GROUP/hpx/issues/794
• Issue #721\(^{3548}\) - Make HPX usable for Xeon Phi
• Issue #694\(^{3549}\) - CMake should complain if you attempt to build an example without its dependencies
• Issue #692\(^{3550}\) - SLURM support broken
• Issue #683\(^{3551}\) - python/hpx/process.py imports epoll on all platforms
• Issue #619\(^{3552}\) - Automate the doc building process
• Issue #600\(^{3553}\) - GTC performance broken
• Issue #577\(^{3554}\) - Allow for zero copy serialization/networking
• Issue #551\(^{3555}\) - Change executable names to have debug postfix in Debug builds
• Issue #544\(^{3556}\) - Write a custom .lib file on Windows pulling in hpx_init and hpx.dll, phase out hpx_init
• Issue #534\(^{3557}\) - hpx::init should take functions by std::function and should accept all forms of hpx_main
• Issue #508\(^{3558}\) - FindPackage fails to set FOO_LIBRARY_DIR
• Issue #506\(^{3559}\) - Add cmake support to generate ini files for external applications
• Issue #470\(^{3560}\) - Changing build-type after configure does not update boost library names
• Issue #453\(^{3561}\) - Document hpx_run_tests.py
• Issue #445\(^{3562}\) - Significant performance mismatch between MPI and HPX in SMP for allgather example
• Issue #443\(^{3563}\) - Make docs viewable from build directory
• Issue #421\(^{3564}\) - Support multiple HPX instances per node in a batch environment like PBS or SLURM
• Issue #316\(^{3565}\) - Add message size limitation
• Issue #249\(^{3566}\) - Clean up locking code in big boot barrier
• Issue #136\(^{3567}\) - Persistent CMake variables need to be marked as cache variables

2.11.14  *HPX V0.9.6 (Jul 30, 2013)*

We have had over 1200 commits since the last release and we have closed roughly 140 tickets (bugs, feature requests, etc.).

\(^{3548}\) https://github.com/STEllAR-GROUP/hpx/issues/721
\(^{3549}\) https://github.com/STEllAR-GROUP/hpx/issues/694
\(^{3550}\) https://github.com/STEllAR-GROUP/hpx/issues/692
\(^{3551}\) https://github.com/STEllAR-GROUP/hpx/issues/683
\(^{3552}\) https://github.com/STEllAR-GROUP/hpx/issues/619
\(^{3553}\) https://github.com/STEllAR-GROUP/hpx/issues/600
\(^{3554}\) https://github.com/STEllAR-GROUP/hpx/issues/577
\(^{3555}\) https://github.com/STEllAR-GROUP/hpx/issues/551
\(^{3556}\) https://github.com/STEllAR-GROUP/hpx/issues/544
\(^{3557}\) https://github.com/STEllAR-GROUP/hpx/issues/534
\(^{3558}\) https://github.com/STEllAR-GROUP/hpx/issues/508
\(^{3559}\) https://github.com/STEllAR-GROUP/hpx/issues/506
\(^{3560}\) https://github.com/STEllAR-GROUP/hpx/issues/470
\(^{3561}\) https://github.com/STEllAR-GROUP/hpx/issues/453
\(^{3562}\) https://github.com/STEllAR-GROUP/hpx/issues/445
\(^{3563}\) https://github.com/STEllAR-GROUP/hpx/issues/443
\(^{3564}\) https://github.com/STEllAR-GROUP/hpx/issues/421
\(^{3565}\) https://github.com/STEllAR-GROUP/hpx/issues/316
\(^{3566}\) https://github.com/STEllAR-GROUP/hpx/issues/249
\(^{3567}\) https://github.com/STEllAR-GROUP/hpx/issues/136
General changes

The major new features in this release are:

- We further consolidated the API exposed by HPX. We aligned our APIs as much as possible with the existing C++11 Standard\[3568\] and related proposals to the C++ standardization committee (such as N3632\[3569\] and N3857\[3570\]).

- We implemented a first version of a distributed AGAS service which essentially eliminates all explicit AGAS network traffic.

- We created a native ibverbs parcelport allowing to take advantage of the superior latency and bandwidth characteristics of Infiniband networks.

- We successfully ported HPX to the Xeon Phi platform.

- Support for the SLURM scheduling system was implemented.

- Major efforts have been dedicated to improving the performance counter framework, numerous new counters were implemented and new APIs were added.

- We added a modular parcel compression system allowing to improve bandwidth utilization (by reducing the overall size of the transferred data).

- We added a modular parcel coalescing system allowing to combine several parcels into larger messages. This reduces latencies introduced by the communication layer.

- Added an experimental executors API allowing to use different scheduling policies for different parts of the code. This API has been modelled after the Standards proposal N3562\[3571\]. This API is bound to change in the future, though.

- Added minimal security support for localities which is enforced on the parcelport level. This support is preliminary and experimental and might change in the future.

- We created a parcelport using low level MPI functions. This is in support of legacy applications which are to be gradually ported and to support platforms where MPI is the only available portable networking layer.

- We added a preliminary and experimental implementation of a tuple-space object which exposes an interface similar to such systems described in the literature (see for instance The Linda Coordination Language\[3572\]).

Bug fixes (closed tickets)

Here is a list of the important tickets we closed for this release. This is again a very long list of newly implemented features and fixed issues.

- Issue #806\[3573\] - make (all) in examples folder does nothing
- Issue #805\[3574\] - Adding the introduction and fixing DOCBOOK dependencies for Windows use
- Issue #804\[3575\] - Add stackless (non-suspendable) thread type
- Issue #803\[3576\] - Create proper serialization support functions for util::tuple
- Issue #800\[3577\] - Add possibility to disable array optimizations during serialization
• Issue #798 - HPX_LIMIT does not work for local dataflow
• Issue #797 - Create a parcelport which uses MPI
• Issue #796 - Problem with Large Numbers of Threads
• Issue #793 - Changing dataflow test case to hang consistently
• Issue #792 - CMake Error
• Issue #791 - Problems with local::dataflow
• Issue #790 - wait_for() doesn’t compile
• Issue #789 - HPX with Intel compiler segfaults
• Issue #788 - Intel compiler support
• Issue #787 - Fixed SFINAEd specializations
• Issue #786 - Memory issues during benchmarking.
• Issue #785 - Create an API allowing to register external threads with HPX
• Issue #784 - util::plugin is throwing an error when a symbol is not found
• Issue #783 - How does hpx::bind work?
• Issue #782 - Added quotes around STRING REPLACE potentially empty arguments
• Issue #781 - Make sure no exceptions propagate into the thread manager
• Issue #780 - Allow arithmetics performance counters to expand its parameters
• Issue #779 - Test case for 778
• Issue #778 - Swapping futures segfaults
• Issue #777 - hpx::lcos::details::when_xxx don’t restore completion handlers
• Issue #776 - Compiler chokes on dataflow overload with launch policy
• Issue #775 - Runtime error with local dataflow (copying futures?)
• Issue #774 - Using local dataflow without explicit namespace
• Issue #773<sup>3601</sup> - Local dataflow with unwrap: functor operators need to be const
• Issue #772<sup>3602</sup> - Allow (remote) actions to return a future
• Issue #771<sup>3603</sup> - Setting HPX_LIMIT gives huge boost MPL errors
• Issue #770<sup>3604</sup> - Add launch policy to (local) dataflow
• Issue #769<sup>3605</sup> - Make compile time configuration information available
• Issue #768<sup>3606</sup> - Const correctness problem in local dataflow
• Issue #767<sup>3607</sup> - Add launch policies to async
• Issue #766<sup>3608</sup> - Mark data structures for optimized (array based) serialization
• Issue #765<sup>3609</sup> - Align hpx::any with N3508: Any Library Proposal (Revision 2)
• Issue #764<sup>3610</sup> - Align hpx::future with newest N3558: A Standardized Representation of Asynchronous Operations
• Issue #762<sup>3611</sup> - added a human readable output for the ping pong example
• Issue #761<sup>3612</sup> - Ambiguous typename when constructing derived component
• Issue #760<sup>3613</sup> - Simple components can not be derived
• Issue #759<sup>3614</sup> - make install doesn’t give a complete install
• Issue #758<sup>3615</sup> - Stack overflow when using locking_hook<>
• Issue #757<sup>3616</sup> - copy paste error; unsupported function overloading
• Issue #756<sup>3617</sup> - GTCX runtime issue in Gordon
• Issue #755<sup>3618</sup> - Papi counters don’t work with reset and evaluate API's
• Issue #753<sup>3619</sup> - cmake bugfix and improved component action docs
• Issue #752<sup>3620</sup> - hpx simple component docs
• Issue #750<sup>3621</sup> - Add hpx::util::any
• Issue #749<sup>3622</sup> - Thread phase counter is not reset
• Issue #748<sup>3623</sup> - Memory performance counter are not registered

<sup>3601</sup> https://github.com/STEllAR-GROUP/hpx/issues/773
<sup>3602</sup> https://github.com/STEllAR-GROUP/hpx/issues/772
<sup>3603</sup> https://github.com/STEllAR-GROUP/hpx/issues/771
<sup>3604</sup> https://github.com/STEllAR-GROUP/hpx/issues/770
<sup>3605</sup> https://github.com/STEllAR-GROUP/hpx/issues/769
<sup>3606</sup> https://github.com/STEllAR-GROUP/hpx/issues/768
<sup>3607</sup> https://github.com/STEllAR-GROUP/hpx/issues/767
<sup>3608</sup> https://github.com/STEllAR-GROUP/hpx/issues/766
<sup>3609</sup> https://github.com/STEllAR-GROUP/hpx/issues/765
<sup>3610</sup> https://github.com/STEllAR-GROUP/hpx/issues/764
<sup>3611</sup> https://github.com/STEllAR-GROUP/hpx/issues/762
<sup>3612</sup> https://github.com/STEllAR-GROUP/hpx/issues/761
<sup>3613</sup> https://github.com/STEllAR-GROUP/hpx/issues/760
<sup>3614</sup> https://github.com/STEllAR-GROUP/hpx/issues/759
<sup>3615</sup> https://github.com/STEllAR-GROUP/hpx/issues/758
<sup>3616</sup> https://github.com/STEllAR-GROUP/hpx/issues/757
<sup>3617</sup> https://github.com/STEllAR-GROUP/hpx/issues/756
<sup>3618</sup> https://github.com/STEllAR-GROUP/hpx/issues/755
<sup>3619</sup> https://github.com/STEllAR-GROUP/hpx/issues/753
<sup>3620</sup> https://github.com/STEllAR-GROUP/hpx/issues/752
<sup>3621</sup> https://github.com/STEllAR-GROUP/hpx/issues/750
<sup>3622</sup> https://github.com/STEllAR-GROUP/hpx/issues/749
<sup>3623</sup> https://github.com/STEllAR-GROUP/hpx/issues/748
• Issue #747 - Create performance counters exposing arithmetic operations
• Issue #745 - apply_callback needs to invoke callback when applied locally
• Issue #744 - CMake fixes
• Issue #743 - Problem Building github version of HPX
• Issue #742 - Remove HPX_STD_BIND
• Issue #741 - assertion ‘px != 0’ failed: HPX(assertion_failure) for low numbers of OS threads
• Issue #739 - Performance counters do not count to the end of the program or evaluation
• Issue #738 - Dedicated AGAS server runs don’t work; console ignores -a option.
• Issue #737 - Missing bind overloads
• Issue #736 - Performance counter wildcards do not always work
• Issue #735 - Create native ibverbs parcelport based on rdma operations
• Issue #734 - Threads stolen performance counter total is incorrect
• Issue #733 - Test benchmarks need to be checked and fixed
• Issue #732 - Build fails with Mac, using mac ports clang-3.3 on latest git branch
• Issue #731 - Add global start/stop API for performance counters
• Issue #730 - Performance counter values are apparently incorrect
• Issue #729 - Unhandled switch
• Issue #728 - Serialization of hpx::util::function between two localities causes seg faults
• Issue #727 - Memory counters on Mac OS X
• Issue #725 - Restore original thread priority on resume
• Issue #724 - Performance benchmarks do not depend on main HPX libraries
• Issue #723 - [teletype]–hpx:nodes=’cat $PBS_NODEFILE‘ works; –hpx:nodefile=$PBS_NODEFILE does not.[c++]
• Issue #722 - Fix binding const member functions as actions

https://github.com/STEllAR-GROUP/hpx/issues/747
https://github.com/STEllAR-GROUP/hpx/issues/745
https://github.com/STEllAR-GROUP/hpx/issues/744
https://github.com/STEllAR-GROUP/hpx/issues/743
https://github.com/STEllAR-GROUP/hpx/issues/742
https://github.com/STEllAR-GROUP/hpx/issues/741
https://github.com/STEllAR-GROUP/hpx/issues/739
https://github.com/STEllAR-GROUP/hpx/issues/738
https://github.com/STEllAR-GROUP/hpx/issues/737
https://github.com/STEllAR-GROUP/hpx/issues/736
https://github.com/STEllAR-GROUP/hpx/issues/735
https://github.com/STEllAR-GROUP/hpx/issues/734
https://github.com/STEllAR-GROUP/hpx/issues/733
https://github.com/STEllAR-GROUP/hpx/issues/732
https://github.com/STEllAR-GROUP/hpx/issues/731
https://github.com/STEllAR-GROUP/hpx/issues/730
https://github.com/STEllAR-GROUP/hpx/issues/729
https://github.com/STEllAR-GROUP/hpx/issues/728
https://github.com/STEllAR-GROUP/hpx/issues/727
https://github.com/STEllAR-GROUP/hpx/issues/724
https://github.com/STEllAR-GROUP/hpx/issues/723
https://github.com/STEllAR-GROUP/hpx/issues/722
• Issue #719 - Create performance counter exposing compression ratio
• Issue #718 - Add possibility to compress parcel data
• Issue #717 - strip_credit_from_gid has misleading semantics
• Issue #716 - Non-option arguments to programs run using pbsdsh must be before --hpnodes, contrary to directions
• Issue #715 - Re-thrown exceptions should retain the original call site
• Issue #714 - failed assertion in debug mode
• Issue #713 - Add performance counters monitoring connection caches
• Issue #712 - Adjust parcel related performance counters to be connection type specific
• Issue #711 - configuration failure
• Issue #710 - Error “timed out while trying to find room in the connection cache” when trying to start multiple localities on a single computer
• Issue #709 - Add new thread state ‘staged’ referring to task descriptions
• Issue #708 - Detect/mitigate bad non-system installs of GCC on Redhat systems
• Issue #707 - Many examples do not link with Git HEAD version
• Issue #706 - hpx::init removes portions of non-option command line arguments before last = sign
• Issue #705 - Create rolling average and median aggregating performance counters
• Issue #704 - Create performance counter to expose thread queue waiting time
• Issue #703 - Add support to HPX build system to find librcrtool.a and related headers
• Issue #699 - Generalize instrumentation support
• Issue #698 - compilation failure with hwloc absent
• Issue #697 - Performance counter counts should be zero indexed
• Issue #696 - Distributed problem
• Issue #695 - Bad perf counter time printed
• Issue #693 - --help doesn’t print component specific command line options
• Issue #692 - SLURM support broken
• Issue #691 - exception while executing any application linked with hwloc
• Issue #690 - thread_id_test and thread_launcher_test failing
• Issue #689 - Make the buildbots use hwloc
• Issue #687 - compilation error fix (hwloc_topology)
• Issue #686 - Linker Error for Applications
• Issue #684 - Pinning of service thread fails when number of worker threads equals the number of cores
• Issue #682 - Add performance counters exposing number of stolen threads
• Issue #681 - Add apply_continue for asynchronous chaining of actions
• Issue #679 - Remove obsolete async_callback API functions
• Issue #678 - Add new API for setting/triggering LCOs
• Issue #677 - Add async_continue for true continuation style actions
• Issue #676 - Buildbot for gcc 4.4 broken
• Issue #675 - Partial preprocessing broken
• Issue #674 - HPX segfaults when built with gcc 4.7
• Issue #673 - use_guard_pages has inconsistent preprocessor guards
• Issue #672 - External build breaks if library path has spaces
• Issue #671 - release tarballs are tarbombs
• Issue #670 - CMake won’t find Boost headers in layout=versioned install
• Issue #669 - Links in docs to source files broken if not installed
• Issue #667 - Not reading ini file properly
• Issue #664 - Adapt new meanings of ‘const’ and ‘mutable’
• Issue #661\footnote{3692} - Implement BTL Parcel port
• Issue #655\footnote{3693} - Make HPX work with the “decltype” result_of
• Issue #647\footnote{3694} - documentation for specifying the number of high priority threads
  --hpx:high-priority-threads
• Issue #643\footnote{3695} - Error parsing host file
• Issue #642\footnote{3696} - HWLoc issue with TAU
• Issue #639\footnote{3697} - Logging potentially suspends a running thread
• Issue #634\footnote{3698} - Improve error reporting from parcel layer
• Issue #627\footnote{3699} - Add tests for async and apply overloads that accept regular C++ functions
• Issue #626\footnote{3700} - hpx/future.hpp header
• Issue #601\footnote{3701} - Intel support
• Issue #557\footnote{3702} - Remove action codes
• Issue #531\footnote{3703} - AGAS request and response classes should use switch statements
• Issue #529\footnote{3704} - Investigate the state of hwloc support
• Issue #526\footnote{3705} - Make HPX aware of hyper-threading
• Issue #518\footnote{3706} - Create facilities allowing to use plain arrays as action arguments
• Issue #473\footnote{3707} - hwloc thread binding is broken on CPUs with hyperthreading
• Issue #383\footnote{3708} - Change result type detection for hpx::util::bind to use result_of protocol
• Issue #341\footnote{3709} - Consolidate route code
• Issue #219\footnote{3710} - Only copy arguments into actions once
• Issue #177\footnote{3711} - Implement distributed AGAS
• Issue #43\footnote{3712} - Support for Darwin (Xcode + Clang)
2.11.15 HPX V0.9.5 (Jan 16, 2013)

We have had over 1000 commits since the last release and we have closed roughly 150 tickets (bugs, feature requests, etc.).

General changes

This release is continuing along the lines of code and API consolidation, and overall usability improvements. We dedicated much attention to performance and we were able to significantly improve the threading and networking subsystems.

We successfully ported HPX to the Android platform. HPX applications now not only can run on mobile devices, but we support heterogeneous applications running across architecture boundaries. At the Supercomputing Conference 2012 we demonstrated connecting Android tablets to simulations running on a Linux cluster. The Android tablet was used to query performance counters from the Linux simulation and to steer its parameters.

We successfully ported HPX to Mac OSX (using the Clang compiler). Thanks to Pyry Jahkola for contributing the corresponding patches. Please see the section How to install HPX on OS X (Mac) for more details.

We made a special effort to make HPX usable in highly concurrent use cases. Many of the HPX API functions which possibly take longer than 100 microseconds to execute now can be invoked asynchronously. We added uniform support for composing futures which simplifies to write asynchronous code. HPX actions (function objects encapsulating possibly concurrent remote function invocations) are now well integrated with all other API facilities such like hpx::bind.

All of the API has been aligned as much as possible with established paradigms. HPX now mirrors many of the facilities as defined in the C++11 Standard, such as hpx::thread, hpx::function, hpx::future, etc.

A lot of work has been put into improving the documentation. Many of the API functions are documented now, concepts are explained in detail, and examples are better described than before. The new documentation index enables finding information with lesser effort.

This is the first release of HPX we perform after the move to Github

Bug fixes (closed tickets)

Here is a list of the important tickets we closed for this release. This is by far the longest list of newly implemented features and fixed issues for any of HPX’ releases so far.

- Issue #666 - Segfault on calling hpx::finalize twice
- Issue #665 - Adding declaration num_of_cores
- Issue #662 - pkgconfig is building wrong
- Issue #660 - Need uninterrupt function
- Issue #659 - Move our logging library into a different namespace
- Issue #658 - Dynamic performance counter types are broken
- Issue #657 - HPX v0.9.5 (RC1) hello_world example segfaulting

References:

3713 https://github.com/STEllAR-GROUP/hpx/
3714 https://www.boost.org/LICENSE_1_0.txt
3715 https://github.com/STEllAR-GROUP/hpx/issues/666
3716 https://github.com/STEllAR-GROUP/hpx/issues/665
3717 https://github.com/STEllAR-GROUP/hpx/issues/662
3718 https://github.com/STEllAR-GROUP/hpx/issues/660
3719 https://github.com/STEllAR-GROUP/hpx/issues/659
3720 https://github.com/STEllAR-GROUP/hpx/issues/658
3721 https://github.com/STEllAR-GROUP/hpx/issues/657

Chapter 2. What’s so special about HPX?
• Issue #656 - Define the affinity of parcel-pool, io-pool, and timer-pool threads
• Issue #654 - Integrate the Boost auto_index tool with documentation
• Issue #653 - Make HPX build on OS X + Clang + libc++
• Issue #651 - Add fine-grained control for thread pinning
• Issue #650 - Command line no error message when using -hpx:(anything)
• Issue #645 - Command line aliases don’t work in [teletype]`@file`[c++]
• Issue #644 - Terminated threads are not always properly cleaned up
• Issue #640 - future_data<T>::set_on_completed_used without locks
• Issue #638 - hpx build with intel compilers fails on linux
• Issue #637 - --copy-dt-needed-entries breaks with gold
• Issue #635 - Boost V1.53 will add Boost.Lockfree and Boost.Atomic
• Issue #633 - Re-add examples to final 0.9.5 release
• Issue #632 - Example thread_aware_timer is broken
• Issue #631 - FFT application throws error in parcellayer
• Issue #630 - Event synchronization example is broken
• Issue #629 - Waiting on futures hangs
• Issue #628 - Add an HPX_ALWAYS_ASSERT macro
• Issue #625 - Port coroutines context switch benchmark
• Issue #621 - New INI section for stack sizes
• Issue #618 - pkg_config support does not work with a HPX debug build
• Issue #617 - hpx/external/logging/boost/logging/detail/cache_before_init.hpp:139:67: error: ‘get_thread_id’ was not declared in this scope
• Issue #616 - Change wait_xxx not to use locking
• Issue #615 - Revert visibility ‘fix’ (fb06b8245dad1127b0c25ebaf9386b3945ca9)

https://github.com/STEllAR-GROUP/hpx/issues/656
https://github.com/STEllAR-GROUP/hpx/issues/654
https://github.com/STEllAR-GROUP/hpx/issues/653
https://github.com/STEllAR-GROUP/hpx/issues/651
https://github.com/STEllAR-GROUP/hpx/issues/650
https://github.com/STEllAR-GROUP/hpx/issues/646
https://github.com/STEllAR-GROUP/hpx/issues/644
https://github.com/STEllAR-GROUP/hpx/issues/640
https://github.com/STEllAR-GROUP/hpx/issues/638
https://github.com/STEllAR-GROUP/hpx/issues/637
https://github.com/STEllAR-GROUP/hpx/issues/635
https://github.com/STEllAR-GROUP/hpx/issues/631
https://github.com/STEllAR-GROUP/hpx/issues/630
https://github.com/STEllAR-GROUP/hpx/issues/629
https://github.com/STEllAR-GROUP/hpx/issues/628
https://github.com/STEllAR-GROUP/hpx/issues/625
https://github.com/STEllAR-GROUP/hpx/issues/621
https://github.com/STEllAR-GROUP/hpx/issues/618
https://github.com/STEllAR-GROUP/hpx/issues/617
https://github.com/STEllAR-GROUP/hpx/issues/616
https://github.com/STEllAR-GROUP/hpx/issues/615

2.11. Releases
• Issue #614\textsuperscript{3745} - Fix Dataflow linker error
• Issue #613\textsuperscript{3746} - find_here should throw an exception on failure
• Issue #612\textsuperscript{3747} - Thread phase doesn’t show up in debug mode
• Issue #611\textsuperscript{3748} - Make stack guard pages configurable at runtime (initialization time)
• Issue #610\textsuperscript{3749} - Co-Locate Components
• Issue #609\textsuperscript{3750} - future\_overhead
• Issue #608\textsuperscript{3751} - --hpx:list-counter-infos problem
• Issue #607\textsuperscript{3752} - Update Boost.Context based backend for coroutines
• Issue #606\textsuperscript{3753} - 1d\_wave\_equation is not working
• Issue #605\textsuperscript{3754} - Any C++ function that has serializable arguments and a serializable return type should be remotable
• Issue #604\textsuperscript{3755} - Connecting localities isn’t working anymore
• Issue #603\textsuperscript{3756} - Do not verify any ini entries read from a file
• Issue #602\textsuperscript{3757} - Rename argument\_size to type\_size/added implementation to get parcel size
• Issue #599\textsuperscript{3758} - Enable locality specific command line options
• Issue #598\textsuperscript{3759} - Need an API that accesses the performance counter reporting the system uptime
• Issue #597\textsuperscript{3760} - compiling on ranger
• Issue #595\textsuperscript{3761} - I need a place to store data in a thread self pointer
• Issue #594\textsuperscript{3762} - 32/64 interoperability
• Issue #593\textsuperscript{3763} - Warn if logging is disabled at compile time but requested at runtime
• Issue #592\textsuperscript{3764} - Add optional argument value to --hpx:list-counters and --hpx:list-counter-infos
• Issue #591\textsuperscript{3765} - Allow for wildcards in performance counter names specified with --hpx:print-counter
• Issue #590\textsuperscript{3766} - Local promise semantic differences

\textsuperscript{3745} https://github.com/STEllAR-GROUP/hpx/issues/614
\textsuperscript{3746} https://github.com/STEllAR-GROUP/hpx/issues/613
\textsuperscript{3747} https://github.com/STEllAR-GROUP/hpx/issues/612
\textsuperscript{3748} https://github.com/STEllAR-GROUP/hpx/issues/611
\textsuperscript{3749} https://github.com/STEllAR-GROUP/hpx/issues/610
\textsuperscript{3750} https://github.com/STEllAR-GROUP/hpx/issues/609
\textsuperscript{3751} https://github.com/STEllAR-GROUP/hpx/issues/608
\textsuperscript{3752} https://github.com/STEllAR-GROUP/hpx/issues/607
\textsuperscript{3753} https://github.com/STEllAR-GROUP/hpx/issues/606
\textsuperscript{3754} https://github.com/STEllAR-GROUP/hpx/issues/605
\textsuperscript{3755} https://github.com/STEllAR-GROUP/hpx/issues/604
\textsuperscript{3756} https://github.com/STEllAR-GROUP/hpx/issues/603
\textsuperscript{3757} https://github.com/STEllAR-GROUP/hpx/issues/602
\textsuperscript{3758} https://github.com/STEllAR-GROUP/hpx/issues/599
\textsuperscript{3759} https://github.com/STEllAR-GROUP/hpx/issues/598
\textsuperscript{3760} https://github.com/STEllAR-GROUP/hpx/issues/597
\textsuperscript{3761} https://github.com/STEllAR-GROUP/hpx/issues/595
\textsuperscript{3762} https://github.com/STEllAR-GROUP/hpx/issues/594
\textsuperscript{3763} https://github.com/STEllAR-GROUP/hpx/issues/593
\textsuperscript{3764} https://github.com/STEllAR-GROUP/hpx/issues/592
\textsuperscript{3765} https://github.com/STEllAR-GROUP/hpx/issues/591
\textsuperscript{3766} https://github.com/STEllAR-GROUP/hpx/issues/590
• Issue #589 - Create API to query performance counter names
• Issue #587 - Add get_num_localities and get_num_threads to AGAS API
• Issue #586 - Adjust local AGAS cache size based on number of localities
• Issue #585 - Error while using counters in HPX
• Issue #584 - counting argument size of actions, initial pass.
• Issue #581 - Remove RemoteResult template parameter for future<>
• Issue #580 - Add possibility to hook into actions
• Issue #578 - Use angle brackets in HPX error dumps
• Issue #576 - Exception incorrectly thrown when --help is used
• Issue #575 - HPX(bad_component_type) with gcc 4.7.2 and boost 1.51
• Issue #574 - --hpx:connect command line parameter not working correctly
• Issue #571 - hpx::wait() (callback version) should pass the future to the callback function
• Issue #570 - hpx::wait should operate on boost::arrays and std::lists
• Issue #569 - Add a logging sink for Android
• Issue #568 - 2-argument version of HPX_DEFINE_COMPONENT_ACTION
• Issue #567 - Connecting to a running HPX application works only once
• Issue #565 - HPX doesn’t shutdown properly
• Issue #564 - Partial preprocessing of new component creation interface
• Issue #563 - Add hpx::start/hpx::stop to avoid blocking main thread
• Issue #562 - All command line arguments swallowed by hpx
• Issue #561 - Boost.Tuple is not move aware
• Issue #558 - boost::shared_ptr<> style semantics/syntax for client classes
• Issue #556 - Creation of partially preprocessed headers should be enabled for Boost newer than V1.50
• Issue #555[^790] - BOOST_FORCEINLINE does not name a type
• Issue #554[^791] - Possible race condition in thread get_id()
• Issue #552[^792] - Move enable client_base
• Issue #550[^793] - Add stack size category ‘huge’
• Issue #549[^794] - ShenEOS run seg-faults on single or distributed runs
• Issue #545[^795] - AUTOGLOB broken for add_hpx_component
• Issue #542[^796] - FindHPX_HDF5 still searches multiple times
• Issue #541[^797] - Quotes around application name in hpx::init
• Issue #539[^798] - Race condition occurring with new lightweight threads
• Issue #535[^799] - hpx_run_tests.py exits with no error code when tests are missing
• Issue #530[^800] - Thread description(<unknown>) in logs
• Issue #523[^801] - Make thread objects more lightweight
• Issue #521[^802] - hpx::error_code is not usable for lightweight error handling
• Issue #520[^803] - Add full user environment to HPX logs
• Issue #519[^804] - Build succeeds, running fails
• Issue #517[^805] - Add a guard page to linux coroutine stacks
• Issue #516[^806] - hpx::thread::detach suspends while holding locks, leads to hang in debug
• Issue #514[^807] - Preprocessed headers for <hpx/apply.hpp> don’t compile
• Issue #513[^808] - Buildbot configuration problem
• Issue #512[^809] - Implement action based stack size customization
• Issue #511[^810] - Move action priority into a separate type trait
• Issue #510[^811] - trunk broken
• Issue #507[^812] - no matching function for call to boost::scoped_ptr<hpx::threads::topology>::scoped_ptr(hpx::threads::linux_topology*)

[^790]: https://github.com/STEllAR-GROUP/hpx/issues/555
[^791]: https://github.com/STEllAR-GROUP/hpx/issues/554
[^792]: https://github.com/STEllAR-GROUP/hpx/issues/552
[^793]: https://github.com/STEllAR-GROUP/hpx/issues/550
[^794]: https://github.com/STEllAR-GROUP/hpx/issues/549
[^795]: https://github.com/STEllAR-GROUP/hpx/issues/545
[^796]: https://github.com/STEllAR-GROUP/hpx/issues/542
[^797]: https://github.com/STEllAR-GROUP/hpx/issues/541
[^798]: https://github.com/STEllAR-GROUP/hpx/issues/539
[^799]: https://github.com/STEllAR-GROUP/hpx/issues/535
[^800]: https://github.com/STEllAR-GROUP/hpx/issues/530
[^801]: https://github.com/STEllAR-GROUP/hpx/issues/523
[^802]: https://github.com/STEllAR-GROUP/hpx/issues/521
[^803]: https://github.com/STEllAR-GROUP/hpx/issues/520
[^804]: https://github.com/STEllAR-GROUP/hpx/issues/519
[^805]: https://github.com/STEllAR-GROUP/hpx/issues/517
[^806]: https://github.com/STEllAR-GROUP/hpx/issues/516
[^807]: https://github.com/STEllAR-GROUP/hpx/issues/514
[^808]: https://github.com/STEllAR-GROUP/hpx/issues/513
[^809]: https://github.com/STEllAR-GROUP/hpx/issues/512
[^810]: https://github.com/STEllAR-GROUP/hpx/issues/511
[^811]: https://github.com/STEllAR-GROUP/hpx/issues/510
[^812]: https://github.com/STEllAR-GROUP/hpx/issues/507
- Issue #505\(^{3813}\) - undefined_symbol regression test currently failing
- Issue #502\(^{3814}\) - Adding OpenCL and OCLM support to HPX for Windows and Linux
- Issue #501\(^{3815}\) - find_package(HPX) sets cmake output variables
- Issue #500\(^{3816}\) - wait_any/wait_all are badly named
- Issue #499\(^{3817}\) - Add support for disabling pbs support in pbs runs
- Issue #498\(^{3818}\) - Error during no-cache runs
- Issue #496\(^{3819}\) - Add partial preprocessing support to cmake
- Issue #495\(^{3820}\) - Support HPX modules exporting startup/shutdown functions only
- Issue #494\(^{3821}\) - Allow modules to specify when to run startup/shutdown functions
- Issue #493\(^{3822}\) - Avoid constructing a string in make_success_code
- Issue #492\(^{3823}\) - Performance counter creation is no longer synchronized at startup
- Issue #491\(^{3824}\) - Performance counter creation is no longer synchronized at startup
- Issue #490\(^{3825}\) - Sheneos on_completed_bulk seg fault in distributed
- Issue #489\(^{3826}\) - compiling issue with g++44
- Issue #488\(^{3827}\) - Adding OpenCL and OCLM support to HPX for the MSVC platform
- Issue #487\(^{3828}\) - FindHPX.cmake problems
- Issue #485\(^{3829}\) - Change distributing_factory and binpacking_factory to use bulk creation

Issue #484\(^{3830}\) - Change HPX\_DON\_T\_USE\_PREPROCESSED\_FILES to

HPX\_USE\_PREPROCESSED\_FILES

- Issue #483\(^{3831}\) - Memory counter for Windows
- Issue #479\(^{3832}\) - strange errors appear when requesting performance counters on multiple nodes
- Issue #477\(^{3833}\) - Create (global) timer for multi-threaded measurements
- Issue #472\(^{3834}\) - Add partial preprocessing using Wave
- Issue #471\(^{3835}\) - Segfault stack traces don’t show up in release

3813 https://github.com/STEllAR-GROUP/hpx/issues/505
3814 https://github.com/STEllAR-GROUP/hpx/issues/502
3815 https://github.com/STEllAR-GROUP/hpx/issues/501
3816 https://github.com/STEllAR-GROUP/hpx/issues/500
3817 https://github.com/STEllAR-GROUP/hpx/issues/499
3818 https://github.com/STEllAR-GROUP/hpx/issues/498
3819 https://github.com/STEllAR-GROUP/hpx/issues/496
3820 https://github.com/STEllAR-GROUP/hpx/issues/495
3821 https://github.com/STEllAR-GROUP/hpx/issues/494
3822 https://github.com/STEllAR-GROUP/hpx/issues/493
3823 https://github.com/STEllAR-GROUP/hpx/issues/492
3824 https://github.com/STEllAR-GROUP/hpx/issues/491
3825 https://github.com/STEllAR-GROUP/hpx/issues/490
3826 https://github.com/STEllAR-GROUP/hpx/issues/489
3827 https://github.com/STEllAR-GROUP/hpx/issues/488
3828 https://github.com/STEllAR-GROUP/hpx/issues/487
3829 https://github.com/STEllAR-GROUP/hpx/issues/485
3830 https://github.com/STEllAR-GROUP/hpx/issues/484
3831 https://github.com/STEllAR-GROUP/hpx/issues/483
3832 https://github.com/STEllAR-GROUP/hpx/issues/479
3833 https://github.com/STEllAR-GROUP/hpx/issues/477
3834 https://github.com/STEllAR-GROUP/hpx/issues/472
3835 https://github.com/STEllAR-GROUP/hpx/issues/471

2.11. Releases
• Issue #468 - External projects need to link with internal components
• Issue #462 - Startup/shutdown functions are called more than once
• Issue #458 - Consolidate hpx::util::high_resolution_timer and hpx::util::high_resolution_clock
• Issue #457 - index out of bounds in allgather_and_gate on 4 cores or more
• Issue #448 - Make HPX compile with clang
• Issue #447 - 'make tests' should execute tests on local installation
• Issue #446 - Remove SVN-related code from the codebase
• Issue #444 - race condition in smp
• Issue #441 - Patched Boost.Serialization headers should only be installed if needed
• Issue #439 - Components using HPX_REGISTER_STARTUP_MODULE fail to compile with MSVC
• Issue #436 - Verify that no locks are being held while threads are suspended
• Issue #435 - Installing HPX should not clobber existing Boost installation
• Issue #434 - Logging external component failed (Boost 1.50)
• Issue #433 - Runtime crash when building all examples
• Issue #432 - Dataflow hangs on 512 cores/64 nodes
• Issue #430 - Problem with distributing factory
• Issue #424 - File paths referring to XSL-files need to be properly escaped
• Issue #417 - Make dataflow LCOs work out of the box by using partial preprocessing
• Issue #413 - hpx_svnversion.py fails on Windows
• Issue #412 - Make hpx::error_code equivalent to hpx::exception
• Issue #398 - HPX clobbers out-of-tree application specific CMake variables (specifically CMAKE_BUILD_TYPE)
• Issue #394 - Remove code generating random port numbers for network
• Issue #378 - ShenEOS scaling issues

https://github.com/STEllAR-GROUP/hpx/issues/468
https://github.com/STEllAR-GROUP/hpx/issues/462
https://github.com/STEllAR-GROUP/hpx/issues/458
https://github.com/STEllAR-GROUP/hpx/issues/457
https://github.com/STEllAR-GROUP/hpx/issues/448
https://github.com/STEllAR-GROUP/hpx/issues/447
https://github.com/STEllAR-GROUP/hpx/issues/446
https://github.com/STEllAR-GROUP/hpx/issues/444
https://github.com/STEllAR-GROUP/hpx/issues/441
https://github.com/STEllAR-GROUP/hpx/issues/439
https://github.com/STEllAR-GROUP/hpx/issues/436
https://github.com/STEllAR-GROUP/hpx/issues/435
https://github.com/STEllAR-GROUP/hpx/issues/434
https://github.com/STEllAR-GROUP/hpx/issues/433
https://github.com/STEllAR-GROUP/hpx/issues/432
https://github.com/STEllAR-GROUP/hpx/issues/430
https://github.com/STEllAR-GROUP/hpx/issues/424
https://github.com/STEllAR-GROUP/hpx/issues/417
https://github.com/STEllAR-GROUP/hpx/issues/413
https://github.com/STEllAR-GROUP/hpx/issues/412
https://github.com/STEllAR-GROUP/hpx/issues/394
https://github.com/STEllAR-GROUP/hpx/issues/378
• Issue #354 - Create a coroutines wrapper for Boost.Context
• Issue #349 - Commandline option --localities=N/\-1N should be necessary only on AGAS locality
• Issue #334 - Add auto_index support to cmake based documentation toolchain
• Issue #318 - Network benchmarks
• Issue #317 - Implement network performance counters
• Issue #310 - Duplicate logging entries
• Issue #230 - Add compile time option to disable thread debugging info
• Issue #171 - Add an INI option to turn off deadlock detection independently of logging
• Issue #170 - OSHL internal counters are incorrect
• Issue #103 - Better diagnostics for multiple component/action registrations under the same name
• Issue #48 - Support for Darwin (Xcode + Clang)
• Issue #21 - Build fails with GCC 4.6

2.11.16 HPX V0.9.0 (Jul 5, 2012)

We have had roughly 800 commits since the last release and we have closed approximately 80 tickets (bugs, feature requests, etc.).

General changes

• Significant improvements made to the usability of HPX in large-scale, distributed environments.
  
  • Renamed hpx::lcos::packaged_task to hpx::lcos::packaged_action to reflect the semantic differences to a packaged_task as defined by the C++11 Standard\(^{3871}\).
  
  • HPX now exposes hpx::thread which is compliant to the C++11 std::thread type except that it (purely locally) represents an HPX thread. This new type does not expose any of the remote capabilities of the underlying HPX-thread implementation.
  
  • The type hpx::lcos::future is now compliant to the C++11 std::future<> type. This type can be used to synchronize both, local and remote operations. In both cases the control flow will `return` to the future in order to trigger any continuation.
  
  • The types hpx::lcos::local::promise and hpx::lcos::local::packaged_task are now compliant to the C++11 std::promise<> and std::packaged_task<> types. These can be used to create a future representing local work only. Use the types hpx::lcos::promise and hpx::lcos::packaged_action to wrap any (possibly remote) action into a future.
  
  • hpx::thread and hpx::lcos::future are now cancelable.

\(^{3859}\) https://github.com/STEllAR-GROUP/hpx/issues/354
\(^{3860}\) https://github.com/STEllAR-GROUP/hpx/issues/349
\(^{3861}\) https://github.com/STEllAR-GROUP/hpx/issues/334
\(^{3862}\) https://github.com/STEllAR-GROUP/hpx/issues/318
\(^{3863}\) https://github.com/STEllAR-GROUP/hpx/issues/317
\(^{3864}\) https://github.com/STEllAR-GROUP/hpx/issues/310
\(^{3865}\) https://github.com/STEllAR-GROUP/hpx/issues/230
\(^{3866}\) https://github.com/STEllAR-GROUP/hpx/issues/171
\(^{3867}\) https://github.com/STEllAR-GROUP/hpx/issues/170
\(^{3868}\) https://github.com/STEllAR-GROUP/hpx/issues/103
\(^{3869}\) https://github.com/STEllAR-GROUP/hpx/issues/48
\(^{3870}\) https://github.com/STEllAR-GROUP/hpx/issues/21
\(^{3871}\) http://www.open-std.org/jtc1/sc22/wg21
• Added support for sequential and logic composition of `hpx::lcos::future`s. The member function `hpx::lcos::future::when` permits futures to be sequentially composed. The helper functions `hpx::wait_all`, `hpx::wait_any`, and `hpx::wait_n` can be used to wait for more than one future at a time.

• **HPX** now exposes `hpx::apply` and `hpx::async` as the preferred way of creating (or invoking) any deferred work. These functions are usable with various types of functions, function objects, and actions and provide a uniform way to spawn deferred tasks.

• **HPX** now utilizes `hpx::util::bind` to (partially) bind local functions and function objects, and also actions. Remote bound actions can have placeholders as well.

• **HPX** continuations are now fully polymorphic. The class `hpx::actions::forwarding_continuation` is an example of how the user can write is own types of continuations. It can be used to execute any function as an continuation of a particular action.

• Reworked the action invocation API to be fully conformant to normal functions. Actions can now be invoked using `hpx::apply`, `hpx::async`, or using the `operator()` implemented on actions. Actions themselves can now be cheaply instantiated as they do not have any members anymore.

• Reworked the lazy action invocation API. Actions can now be directly bound using `hpx::util::bind` by passing an action instance as the first argument.

• A minimal **HPX** program now looks like this:

```cpp
#include <hpx/hpx_init.hpp>

int hpx_main()
{
    return hpx::finalize();
}

int main()
{
    return hpx::init();
}
```

This removes the immediate dependency on the Boost.Program Options library.

**Note:** This minimal version of an **HPX** program does not support any of the default command line arguments (such as --help, or command line options related to PBS). It is suggested to always pass `argc` and `argv` to **HPX** as shown in the example below.

• In order to support those, but still not to depend on Boost.Program Options, the minimal program can be written as:

```cpp
#include <hpx/hpx_init.hpp>

// The arguments for hpx_main can be left off, which very similar to the
// behavior of `main()` as defined by C++.
int hpx_main(int argc, char* argv[])
{
    return hpx::finalize();
}
```

(continues on next page)
```c
int main(int argc, char* argv[])
{
    return hpx::init(argc, argv);
}
```

- Added performance counters exposing the number of component instances which are alive on a given locality.
- Added performance counters exposing then number of messages sent and received, the number of parcels sent and received, the number of bytes sent and received, the overall time required to send and receive data, and the overall time required to serialize and deserialize the data.
- Added a new component: `hpx::components::binpacking_factory` which is equivalent to the existing `hpx::components::distributing_factory` component, except that it equalizes the overall population of the components to create. It exposes two factory methods, one based on the number of existing instances of the component type to create, and one based on an arbitrary performance counter which will be queried for all relevant localities.
- Added API functions allowing to access elements of the diagnostic information embedded in the given exception: `hpx::get_locality_id`, `hpx::get_host_name`, `hpx::get_process_id`, `hpx::get_function_name`, `hpx::get_file_name`, `hpx::get_line_number`, `hpx::get_os_thread`, `hpx::get_thread_id`, and `hpx::get_thread_description`.

**Bug fixes (closed tickets)**

Here is a list of the important tickets we closed for this release:

- **Issue #71**[^71] - GIDs that are not serialized via `handle_gid<>` should raise an exception
- **Issue #105**[^105] - Allow for `hpx::util::functions` to be registered in the AGAS symbolic namespace
- **Issue #107**[^107] - Nasty threadmanger race condition (reproducible in sheneos_test)
- **Issue #108**[^108] - Add millisecond resolution to HPX logs on Linux
- **Issue #110**[^110] - Shutdown hang in distributed with release build
- **Issue #116**[^116] - Don’t use TSS for the applier and runtime pointers
- **Issue #162**[^162] - Move local synchronous execution shortcut from `hpx::function` to the applier
- **Issue #172**[^172] - Cache sources in CMake and check if they change manually
- **Issue #178**[^178] - Add an INI option to turn off ranged-based AGAS caching
- **Issue #187**[^187] - Support for disabling performance counter deployment
- **Issue #202**[^202] - Support for sending performance counter data to a specific file
- **Issue #218**[^218] - boost.coroutines allows different stack sizes, but stack pool is unaware of this

[^71]: https://github.com/STEllAR-GROUP/hpx/issues/71
[^105]: https://github.com/STEllAR-GROUP/hpx/issues/105
[^107]: https://github.com/STEllAR-GROUP/hpx/issues/107
[^110]: https://github.com/STEllAR-GROUP/hpx/issues/110
[^116]: https://github.com/STEllAR-GROUP/hpx/issues/116
[^162]: https://github.com/STEllAR-GROUP/hpx/issues/162
[^172]: https://github.com/STEllAR-GROUP/hpx/issues/172
[^178]: https://github.com/STEllAR-GROUP/hpx/issues/178
[^187]: https://github.com/STEllAR-GROUP/hpx/issues/187
[^202]: https://github.com/STEllAR-GROUP/hpx/issues/202
[^218]: https://github.com/STEllAR-GROUP/hpx/issues/218

2.11. Releases
• Issue #231 - Implement movable boost::bind
• Issue #232 - Implement movable boost::function
• Issue #236 - Allow binding hpx::util::function to actions
• Issue #239 - Replace hpx::function with hpx::util::function
• Issue #240 - Can’t specify RemoteResult with lcos::async
• Issue #242 - REGISTER_TEMPLATE support for plain actions
• Issue #243 - handle_gid<> support for hpx::util::function
• Issue #245 - _c_cache code throws an exception if the queried GID is not in the local cache
• Issue #246 - Undefined references in dataflow/adaptive1d example
• Issue #252 - Problems configuring sheneos with CMake
• Issue #254 - Lifetime of components doesn’t end when client goes out of scope
• Issue #259 - CMake does not detect that MSVC10 has lambdas
• Issue #260 - io_service_pool segfault
• Issue #261 - Late parcel executed outside of pxthread
• Issue #263 - Cannot select allocator with CMake
• Issue #264 - Fix allocator select
• Issue #267 - Runtime error for hello_world
• Issue #269 - pthread_affinity_np test fails to compile
• Issue #270 - Compiler noise due to -Wcast-qual
• Issue #275 - Problem with configuration tests/include paths on Gentoo
• Issue #325 - Sheneos is 200-400 times slower than the fortran equivalent
• Issue #331 - hpx::init and hpx_main() should not depend on program_options
• Issue #333 - Add doxygen support to CMake for doc toolchain

3886 https://github.com/STEllAR-GROUP/hpx/issues/231
3887 https://github.com/STEllAR-GROUP/hpx/issues/232
3888 https://github.com/STEllAR-GROUP/hpx/issues/236
3889 https://github.com/STEllAR-GROUP/hpx/issues/239
3890 https://github.com/STEllAR-GROUP/hpx/issues/240
3891 https://github.com/STEllAR-GROUP/hpx/issues/242
3892 https://github.com/STEllAR-GROUP/hpx/issues/243
3893 https://github.com/STEllAR-GROUP/hpx/issues/245
3894 https://github.com/STEllAR-GROUP/hpx/issues/246
3895 https://github.com/STEllAR-GROUP/hpx/issues/252
3896 https://github.com/STEllAR-GROUP/hpx/issues/254
3897 https://github.com/STEllAR-GROUP/hpx/issues/259
3898 https://github.com/STEllAR-GROUP/hpx/issues/260
3899 https://github.com/STEllAR-GROUP/hpx/issues/261
3900 https://github.com/STEllAR-GROUP/hpx/issues/263
3901 https://github.com/STEllAR-GROUP/hpx/issues/264
3902 https://github.com/STEllAR-GROUP/hpx/issues/267
3903 https://github.com/STEllAR-GROUP/hpx/issues/269
3904 https://github.com/STEllAR-GROUP/hpx/issues/270
3905 https://github.com/STEllAR-GROUP/hpx/issues/275
3906 https://github.com/STEllAR-GROUP/hpx/issues/325
3907 https://github.com/STEllAR-GROUP/hpx/issues/331
3908 https://github.com/STEllAR-GROUP/hpx/issues/333
• Issue #340 - Performance counters for parcels
• Issue #346 - Component loading error when running hello_world in distributed on MSVC2010
• Issue #362 - Missing initializer error
• Issue #363 - Parcel port serialization error
• Issue #366 - Parcel buffering leads to types incompatible exception
• Issue #368 - Scalable alternative to rand() needed for HPX
• Issue #369 - IB over IP is substantially slower than just using standard TCP/IP
• Issue #374 - hpx::lcos::wait should work with dataflows and arbitrary classes meeting the future interface
• Issue #375 - Conflicting/ambiguous overloads of hpx::lcos::wait
• Issue #376 - Find_HPX.cmake should set CMake variable HPX_FOUND for out of tree builds
• Issue #377 - ShenEOS interpolate bulk and interpolate_one_bulk are broken
• Issue #379 - Add support for distributed runs under SLURM
• Issue #382 - _Unwind_Word not declared in boost.backtrace
• Issue #387 - Doxygen should look only at list of specified files
• Issue #388 - Running make install on an out-of-tree application is broken
• Issue #391 - Out-of-tree application segfaults when running in qsub
• Issue #392 - Remove HPX_NO_INSTALL option from cmake build system
• Issue #396 - Pragma related warnings when compiling with older gcc versions
• Issue #399 - Out of tree component build problems
• Issue #400 - Out of source builds on Windows: linker should not receive compiler flags
• Issue #401 - Out of source builds on Windows: components need to be linked with hpx_serialization
• Issue #404 - gfortran fails to link automatically when fortran files are present
• Issue #405 - Inability to specify linking order for external libraries
• Issue #4063932 - Adapt action limits such that dataflow applications work without additional defines
• Issue #4153933 - locality_results is not a member of hpx::components::server
• Issue #4253934 - Breaking changes to traits::result wrt std::vector<id_type>
• Issue #4263935 - AUTOGLOB needs to be updated to support fortran

2.11.17  HPX V0.8.1 (Apr 21, 2012)

This is a point release including important bug fixes for HPX V0.8.0 (Mar 23, 2012).

General changes

• HPX does not need to be installed anymore to be functional.

Bug fixes (closed tickets)

Here is a list of the important tickets we closed for this point release:

• Issue #2953936 - Don’t require install path to be known at compile time.
• Issue #3713937 - Add hpx iostreams to standard build.
• Issue #3843938 - Fix compilation with GCC 4.7.
• Issue #3903939 - Remove keep_factory_alive startup call from ShenEOS; add shutdown call to H5close.
• Issue #3933940 - Thread affinity control is broken.

Bug fixes (commits)

Here is a list of the important commits included in this point release:

• r7642 - External: Fix backtrace memory violation.

• r7775 - Components: Fix symbol visibility bug with component startup providers. This prevents one components providers from overriding another components.

• r7778 - Components: Fix startup/shutdown provider shadowing issues.

2.11.18  HPX V0.8.0 (Mar 23, 2012)

We have had roughly 1000 commits since the last release and we have closed approximately 70 tickets (bugs, feature requests, etc.).

General changes

• Improved PBS support, allowing for arbitrary naming schemes of node-hostnames.

• Finished verification of the reference counting framework.

• Implemented decrement merging logic to optimize the distributed reference counting system.

3932 https://github.com/STEllAR-GROUP/hpx/issues/406
3933 https://github.com/STEllAR-GROUP/hpx/issues/415
3934 https://github.com/STEllAR-GROUP/hpx/issues/425
3935 https://github.com/STEllAR-GROUP/hpx/issues/426
3936 https://github.com/STEllAR-GROUP/hpx/issues/295
3937 https://github.com/STEllAR-GROUP/hpx/issues/371
3938 https://github.com/STEllAR-GROUP/hpx/issues/384
3939 https://github.com/STEllAR-GROUP/hpx/issues/390
3940 https://github.com/STEllAR-GROUP/hpx/issues/393
• Restructured the LCO framework. Renamed `hpx::lcos::eager_future<>` and `hpx::lcos::lazy_future<>` into `hpx::lcos::packaged_task` and `hpx::lcos::deferred_packaged_task`. Split `hpx::lcos::promise` into `hpx::lcos::packaged_task` and `hpx::lcos::future`. Added ‘local’ futures (in namespace `hpx::lcos::local`).

• Improved the general performance of local and remote action invocations. This (under certain circumstances) drastically reduces the number of copies created for each of the parameters and return values.

• Reworked the performance counter framework. Performance counters are now created only when needed, which reduces the overall resource requirements. The new framework allows for much more flexible creation and management of performance counters. The new sine example application demonstrates some of the capabilities of the new infrastructure.

• Added a buildbot-based continuous build system which gives instant, automated feedback on each commit to SVN.

• Added more automated tests to verify proper functioning of HPX.

• Started to create documentation for HPX and its API.

• Added documentation toolchain to the build system.

• Added dataflow LCO.

• Changed default HPX command line options to have `hpx:` prefix. For instance, the former option `--threads` is now `--hpx:threads`. This has been done to make ambiguities with possible application specific command line options as unlikely as possible. See the section `HPX Command Line Options` for a full list of available options.

• Added the possibility to define command line aliases. The former short (one-letter) command line options have been predefined as aliases for backwards compatibility. See the section `HPX Command Line Options` for a detailed description of command line option aliasing.

• Network connections are now cached based on the connected host. The number of simultaneous connections to a particular host is now limited. Parcels are buffered and bundled if all connections are in use.

• Added more refined thread affinity control. This is based on the external library Portable Hardware Locality (HWLOC).

• Improved support for Windows builds with CMake.

• Added support for components to register their own command line options.

• Added the possibility to register custom startup/shutdown functions for any component. These functions are guaranteed to be executed by an HPX thread.

• Added two new experimental thread schedulers: hierarchy_scheduler and periodic_priority_scheduler. These can be activated by using the command line options `--hpx:queuing=hierarchy` or `--hpx:queuing=periodic`.

Example applications

• Graph500 performance benchmark\(^3\)\(^9\)\(^4\)\(^1\) (thanks to Matthew Anderson for contributing this application).

• GTC (Gyrokinetic Toroidal Code)\(^3\)\(^9\)\(^4\)\(^2\): a skeleton for particle in cell type codes.

• Random Memory Access: an example demonstrating random memory accesses in a large array.

---

\(^3\)\(^9\)\(^4\)\(^1\) [http://www.graph500.org/](http://www.graph500.org/)

• ShenEOS example\textsuperscript{3943}, demonstrating partitioning of large read-only data structures and exposing an interpolation API.

• Sine performance counter demo.

• Accumulator examples demonstrating how to write and use HPX components.

• Quickstart examples (like hello_world, fibonacci, quicksort, factorial, etc.) demonstrating simple HPX concepts which introduce some of the concepts in HPX.

• Load balancing and work stealing demos.

API changes

• Moved all local LCOs into a separate namespace `hpx::lcos::local` (for instance, `hpx::lcos::local_mutex` is now `hpx::lcos::local::mutex`).

• Replaced `hpx::actions::function` with `hpx::util::function`. Cleaned up related code.

• Removed `hpx::traits::handle_gid` and moved handling of global reference counts into the corresponding serialization code.

• Changed terminology: `prefix` is now called `locality_id`, renamed the corresponding API functions (such as `hpx::get_prefix`, which is now called `hpx::get_locality_id`).

• Adding `hpx::find_remote_localities`, and `hpx::get_num_localities`.

• Changed performance counter naming scheme to make it more bash friendly. The new performance counter naming scheme is now

\[
/\text{object}\{\text{parentname}\#\text{parentindex}\}/\text{instance}\#\text{index}/\text{counter}\#\text{parameters}
\]

• Added `hpx::get_worker_thread_num` replacing `hpx::threadmanager_base::get_thread_num`.

• Renamed `hpx::get_num_os_threads` to `hpx::get_os_threads_count`.

• Added `hpx::threads::get_thread_count`.

• Restructured the Futures sub-system, renaming types in accordance with the terminology used by the C++11 ISO standard.

Bug fixes (closed tickets)

Here is a list of the important tickets we closed for this release:

• Issue #31\textsuperscript{3944} - Specialize handle_gid<> for examples and tests

• Issue #72\textsuperscript{3945} - Fix AGAS reference counting

• Issue #104\textsuperscript{3946} - heartbeat throws an exception when decrefing the performance counter it’s watching

• Issue #111\textsuperscript{3947} - throttle causes an exception on the target application

• Issue #142\textsuperscript{3948} - One failed component loading causes an unrelated component to fail

• Issue #165\textsuperscript{3949} - Remote exception propagation bug in AGAS reference counting test

\textsuperscript{3943} http://stellarcollapse.org/equationofstate
\textsuperscript{3944} https://github.com/STEllAR-GROUP/hpx/issues/31
\textsuperscript{3945} https://github.com/STEllAR-GROUP/hpx/issues/72
\textsuperscript{3946} https://github.com/STEllAR-GROUP/hpx/issues/104
\textsuperscript{3947} https://github.com/STEllAR-GROUP/hpx/issues/111
\textsuperscript{3948} https://github.com/STEllAR-GROUP/hpx/issues/142
\textsuperscript{3949} https://github.com/STEllAR-GROUP/hpx/issues/165
• Issue #186 - Test credit exhaustion/splitting (e.g. prepare_gid and symbol NS)
• Issue #188 - Implement remaining AGAS reference counting test cases
• Issue #258 - No type checking of GIDs in stubs classes
• Issue #271 - Seg fault/shared pointer assertion in distributed code
• Issue #281 - CMake options need descriptive text
• Issue #283 - AGAS caching broken (gva_cache needs to be rewritten with ICL)
• Issue #285 - HPX_INSTALL root directory not the same as CMAKE_INSTALL_PREFIX
• Issue #286 - New segfault in dataflow applications
• Issue #289 - Exceptions should only be logged if not handled
• Issue #290 - c++11 tests failure
• Issue #293 - Build target for component libraries
• Issue #296 - Compilation error with Boost V1.49rc1
• Issue #298 - Illegal instructions on termination
• Issue #299 - gravity aborts with multiple threads
• Issue #301 - Build error with Boost trunk
• Issue #303 - Logging assertion failure in distributed runs
• Issue #304 - Exception ‘what’ strings are lost when exceptions from decode_parcel are reported
• Issue #306 - Performance counter user interface issues
• Issue #307 - Logging exception in distributed runs
• Issue #308 - Logging deadlocks in distributed
• Issue #309 - Reference counting test failures and exceptions
• Issue #311 - Merge AGAS remote_interface with the runtime_support object
• Issue #314 - Object tracking for id_types
• Issue #315\textsuperscript{3973} - Remove handle_gid and handle credit splitting in id_type serialization
• Issue #320\textsuperscript{3974} - applier::get_locality_id() should return an error value (or throw an exception)
• Issue #321\textsuperscript{3975} - Optimization for id_types which are never split should be restored
• Issue #322\textsuperscript{3976} - Command line processing ignored with Boost 1.47.0
• Issue #323\textsuperscript{3977} - Credit exhaustion causes object to stay alive
• Issue #324\textsuperscript{3978} - Duplicate exception messages
• Issue #326\textsuperscript{3979} - Integrate Quickbook with CMake
• Issue #329\textsuperscript{3980} - --help and --version should still work
• Issue #330\textsuperscript{3981} - Create pkg-config files
• Issue #337\textsuperscript{3982} - Improve usability of performance counter timestamps
• Issue #338\textsuperscript{3983} - Non-std exceptions deriving from std::exceptions in tfunc may be sliced
• Issue #339\textsuperscript{3984} - Decrease the number of send_pending_parcel threads
• Issue #343\textsuperscript{3985} - Dynamically setting the stack size doesn’t work
• Issue #351\textsuperscript{3986} - ‘make install’ does not update documents
• Issue #353\textsuperscript{3987} - Disable FIXMEs in the docs by default; add a doc developer CMake option to enable FIXMEs
• Issue #355\textsuperscript{3988} - ‘make’ doesn’t do anything after correct configuration
• Issue #356\textsuperscript{3989} - Don’t use hpx::util::static_ in topology code
• Issue #359\textsuperscript{3990} - Infinite recursion in hpx::tuple serialization
• Issue #361\textsuperscript{3991} - Add compile time option to disable logging completely
• Issue #364\textsuperscript{3992} - Installation seriously broken in r7443

2.11.19  \textit{HPX} V0.7.0 (Dec 12, 2011)

We have had roughly 1000 commits since the last release and we have closed approximately 120 tickets (bugs, feature requests, etc.).

\textsuperscript{3973} https://github.com/STEllAR-GROUP/hpx/issues/315
\textsuperscript{3974} https://github.com/STEllAR-GROUP/hpx/issues/320
\textsuperscript{3975} https://github.com/STEllAR-GROUP/hpx/issues/321
\textsuperscript{3976} https://github.com/STEllAR-GROUP/hpx/issues/322
\textsuperscript{3977} https://github.com/STEllAR-GROUP/hpx/issues/323
\textsuperscript{3978} https://github.com/STEllAR-GROUP/hpx/issues/324
\textsuperscript{3979} https://github.com/STEllAR-GROUP/hpx/issues/326
\textsuperscript{3980} https://github.com/STEllAR-GROUP/hpx/issues/329
\textsuperscript{3981} https://github.com/STEllAR-GROUP/hpx/issues/330
\textsuperscript{3982} https://github.com/STEllAR-GROUP/hpx/issues/337
\textsuperscript{3983} https://github.com/STEllAR-GROUP/hpx/issues/338
\textsuperscript{3984} https://github.com/STEllAR-GROUP/hpx/issues/339
\textsuperscript{3985} https://github.com/STEllAR-GROUP/hpx/issues/343
\textsuperscript{3986} https://github.com/STEllAR-GROUP/hpx/issues/351
\textsuperscript{3987} https://github.com/STEllAR-GROUP/hpx/issues/353
\textsuperscript{3988} https://github.com/STEllAR-GROUP/hpx/issues/355
\textsuperscript{3989} https://github.com/STEllAR-GROUP/hpx/issues/356
\textsuperscript{3990} https://github.com/STEllAR-GROUP/hpx/issues/359
\textsuperscript{3991} https://github.com/STEllAR-GROUP/hpx/issues/361
\textsuperscript{3992} https://github.com/STEllAR-GROUP/hpx/issues/364
General changes

- Completely removed code related to deprecated AGAS V1, started to work on AGAS V2.1.
- Started to clean up and streamline the exposed APIs (see ‘API changes’ below for more details).
- Revamped and unified performance counter framework, added a lot of new performance counter instances for monitoring of a diverse set of internal HPX parameters (queue lengths, access statistics, etc.).
- Improved general error handling and logging support.
- Fixed several race conditions, improved overall stability, decreased memory footprint, improved overall performance (major optimizations include native TLS support and ranged-based AGAS caching).
- Added support for running HPX applications with PBS.
- Many updates to the build system, added support for gcc 4.5.x and 4.6.x, added C++11 support.
- Many updates to default command line options.
- Added many tests, set up buildbot for continuous integration testing.
- Better shutdown handling of distributed applications.

Example applications

- quickstart/factorial and quickstart/fibonacci, future-recursive parallel algorithms.
- quickstart/hello_world, distributed hello world example.
- quickstart/rma, simple remote memory access example
- quickstart/quicksort, parallel quicksort implementation.
- gtc, gyrokinetic torodial code.
- bfs, breadth-first-search, example code for a graph application.
- sheneos, partitioning of large data sets.
- accumulator, simple component example.
- balancing/os_thread_num, balancing/px_thread_phase, examples demonstrating load balancing and work stealing.

API changes

- Added `hpx::find_all_localities`.
- Added `hpx::terminate` for non-graceful termination of applications.
- Added `hpx::lcos::async` functions for simpler asynchronous programming.
- Added new AGAS interface for handling of symbolic namespace (`hpx::agas::*`).
- Renamed `hpx::components::wait` to `hpx::lcos::wait`.
- Renamed `hpx::lcos::future_value` to `hpx::lcos::promise`.
- Renamed `hpx::lcos::recursive_mutex` to `hpx::lcos::local_recursive_mutex`, `hpx::lcos::mutex` to `hpx::lcos::local_mutex`
- Removed support for Boost versions older than V1.38, recommended Boost version is now V1.47 and newer.
- Removed `hpx::process` (this will be replaced by a real process implementation in the future).
- Removed non-functional LCO code (`hpx::lcos::dataflow`, `hpx::lcos::thunk`, `hpx::lcos::dataflow_variable`).
- Removed deprecated hpx::naming::full_address.

**Bug fixes (closed tickets)**

Here is a list of the important tickets we closed for this release:

- **Issue #28** [3993](https://github.com/STEllAR-GROUP/hpx/issues/28) - Integrate Windows/Linux CMake code for HPX core
- **Issue #32** [3994](https://github.com/STEllAR-GROUP/hpx/issues/32) - hpx::cout() should be hpx::cout
- **Issue #33** [3995](https://github.com/STEllAR-GROUP/hpx/issues/33) - AGAS V2 legacy client does not properly handle error_code
- **Issue #60** [3996](https://github.com/STEllAR-GROUP/hpx/issues/60) - AGAS: allow for registerid to optionally take ownership of the gid
- **Issue #62** [3997](https://github.com/STEllAR-GROUP/hpx/issues/62) - adaptive1d compilation failure in Fusion
- **Issue #64** [3998](https://github.com/STEllAR-GROUP/hpx/issues/64) - Parcel subsystem doesn’t resolve domain names
- **Issue #83** [3999](https://github.com/STEllAR-GROUP/hpx/issues/83) - No error handling if no console is available
- **Issue #84** [4000](https://github.com/STEllAR-GROUP/hpx/issues/84) - No error handling if a hosted locality is treated as the bootstrap server
- **Issue #90** [4001](https://github.com/STEllAR-GROUP/hpx/issues/90) - Add general commandline option -N
- **Issue #91** [4002](https://github.com/STEllAR-GROUP/hpx/issues/91) - Add possibility to read command line arguments from file
- **Issue #92** [4003](https://github.com/STEllAR-GROUP/hpx/issues/92) - Always log exceptions/errors to the log file
- **Issue #93** [4004](https://github.com/STEllAR-GROUP/hpx/issues/93) - Log the command line/program name
- **Issue #95** [4005](https://github.com/STEllAR-GROUP/hpx/issues/95) - Support for distributed launches
- **Issue #97** [4006](https://github.com/STEllAR-GROUP/hpx/issues/97) - Attempt to create a bad component type in AMR examples
- **Issue #100** [4007](https://github.com/STEllAR-GROUP/hpx/issues/100) - factorial and factorial_get examples trigger AGAS component type assertions
- **Issue #101** [4008](https://github.com/STEllAR-GROUP/hpx/issues/101) - Segfault when hpx::process::here() is called in fibonacci2
- **Issue #102** [4009](https://github.com/STEllAR-GROUP/hpx/issues/102) - unknown_component_address in int_object_semaphore_client
- **Issue #114** [4010](https://github.com/STEllAR-GROUP/hpx/issues/114) - marduk raises assertion with default parameters
- **Issue #115** [4011](https://github.com/STEllAR-GROUP/hpx/issues/115) - Logging messages for SMP runs (on the console) shouldn’t be buffered
- **Issue #119** [4012](https://github.com/STEllAR-GROUP/hpx/issues/119) - marduk linking strategy breaks other applications
- **Issue #121** [4013](https://github.com/STEllAR-GROUP/hpx/issues/121) - pbsdsh problem
• Issue #123\footnote{https://github.com/STEllAR-GROUP/hpx/issues/123} - marduk, dataflow and adaptive1d fail to build
• Issue #124\footnote{https://github.com/STEllAR-GROUP/hpx/issues/124} - Lower default preprocessing arity
• Issue #125\footnote{https://github.com/STEllAR-GROUP/hpx/issues/125} - Move hpx::detail::diagnostic_information out of the detail namespace
• Issue #126\footnote{https://github.com/STEllAR-GROUP/hpx/issues/126} - Test definitions for AGAS reference counting
• Issue #128\footnote{https://github.com/STEllAR-GROUP/hpx/issues/128} - Add averaging performance counter
• Issue #129\footnote{https://github.com/STEllAR-GROUP/hpx/issues/129} - Error with endian.hpp while building adaptive1d
• Issue #130\footnote{https://github.com/STEllAR-GROUP/hpx/issues/130} - Bad initialization of performance counters
• Issue #131\footnote{https://github.com/STEllAR-GROUP/hpx/issues/131} - Add global startup/shutdown functions to component modules
• Issue #132\footnote{https://github.com/STEllAR-GROUP/hpx/issues/132} - Avoid using auto_ptr
• Issue #133\footnote{https://github.com/STEllAR-GROUP/hpx/issues/133} - On Windows hpx.dll doesn’t get installed
• Issue #134\footnote{https://github.com/STEllAR-GROUP/hpx/issues/134} - HPX_LIBRARY does not reflect real library name (on Windows)
• Issue #135\footnote{https://github.com/STEllAR-GROUP/hpx/issues/135} - Add detection of unique_ptr to build system
• Issue #137\footnote{https://github.com/STEllAR-GROUP/hpx/issues/137} - Add command line option allowing to repeatedly evaluate performance counters
• Issue #139\footnote{https://github.com/STEllAR-GROUP/hpx/issues/139} - Logging is broken
• Issue #140\footnote{https://github.com/STEllAR-GROUP/hpx/issues/140} - CMake problem on windows
• Issue #141\footnote{https://github.com/STEllAR-GROUP/hpx/issues/141} - Move all non-component libraries into SPREFIX/lib/hpx
• Issue #143\footnote{https://github.com/STEllAR-GROUP/hpx/issues/143} - adaptive1d throws an exception with the default command line options
• Issue #146\footnote{https://github.com/STEllAR-GROUP/hpx/issues/146} - Early exception handling is broken
• Issue #147\footnote{https://github.com/STEllAR-GROUP/hpx/issues/147} - Sheneos doesn’t link on Linux
• Issue #149\footnote{https://github.com/STEllAR-GROUP/hpx/issues/149} - sheneos_test hangs
• Issue #154\footnote{https://github.com/STEllAR-GROUP/hpx/issues/154} - Compilation fails for r5661
• Issue #155\footnote{https://github.com/STEllAR-GROUP/hpx/issues/155} - Sine performance counters example chokes on chrono headers
• Issue #156\footnote{https://github.com/STEllAR-GROUP/hpx/issues/156} - Add build type to –version
• Issue #157 - Extend AGAS caching to store gid ranges
• Issue #158 - r5691 doesn’t compile
• Issue #160 - Re-add AGAS function for resolving a locality to its prefix
• Issue #168 - Managed components should be able to access their own GID
• Issue #169 - Rewrite AGAS future pool
• Issue #179 - Complete switch to request class for AGAS server interface
• Issue #182 - Sine performance counter is loaded by other examples
• Issue #185 - Write tests for symbol namespace reference counting
• Issue #191 - Assignment of read-only variable in point_geometry
• Issue #200 - Seg faults when querying performance counters
• Issue #204 - --ifnames and suffix stripping needs to be more generic
• Issue #205 - --list-* and --print-counter-* options do not work together and produce no warning
• Issue #207 - Implement decrement entry merging
• Issue #208 - Replace the spinlocks in AGAS with hpx::lcos::local_mutexes
• Issue #210 - Add an --ifprefix option
• Issue #214 - Performance test for PX-thread creation
• Issue #216 - VS2010 compilation
• Issue #222 - r6045 context_linux_x86.hpp
• Issue #223 - fibonacci hangs when changing the state of an active thread
• Issue #225 - Active threads end up in the FEB wait queue
• Issue #226 - VS Build Error for Accumulator Client
• Issue #228 - Move all traits into namespace hpx::traits
• Issue #229 - Invalid initialization of reference in thread_init_data
• Issue #235 - Invalid GID in iostreams
• Issue #238 - Demangle type names for the default implementation of get_action_name
• Issue #241 - C++11 support breaks GCC 4.5
• Issue #247 - Reference to temporary with GCC 4.4
• Issue #248 - Seg fault at shutdown with GCC 4.4
• Issue #253 - Default component action registration kills compiler
• Issue #272 - G++ unrecognized command line option
• Issue #273 - quicksort example doesn’t compile
• Issue #277 - Invalid CMake logic for Windows

2.12 About HPX

2.12.1 History

The development of High Performance ParalleX (HPX) began in 2007. At that time, Hartmut Kaiser became interested in the work done by the ParalleX group at the Center for Computation and Technology (CCT), a multi-disciplinary research institute at Louisiana State University (LSU). The ParalleX group was working to develop a new and experimental execution model for future high performance computing architectures. This model was christened ParalleX. The first implementations of ParalleX were crude, and many of those designs had to be discarded entirely. However, over time the team learned quite a bit about how to design a parallel, distributed runtime system which implements the concepts of ParalleX.

From the very beginning, this endeavour has been a group effort. In addition to a handful of interested researchers, there have always been graduate and undergraduate students participating in the discussions, design, and implementation of HPX. In 2011 we decided to formalize our collective research efforts by creating the STEllAR group (Systems Technology, Emergent Parallelism, and Algorithm Research). Over time, the team grew to include researchers around the country and the world. In 2014, the STEllAR Group was reorganized to become the international community it is today. This consortium of researchers aims to develop stable, sustainable, and scalable tools which will enable application developers to exploit the parallelism latent in the machines of today and tomorrow. Our goal of the HPX project is to create a high quality, freely available, open source implementation of ParalleX concepts for conventional and future systems by building a modular and standards conforming runtime system for SMP and distributed application environments. The API exposed by HPX is conformant to the interfaces defined by the C++11/14 ISO standard and adheres to the programming guidelines used by the Boost collection of C++ libraries.

We steer the development of HPX with real world applications and aim to provide a smooth migration path for domain scientists.

To learn more about STEllAR and ParalleX, see People and Why HPX?

4060 https://github.com/STEllAR-GROUP/hpx/issues/235
4061 https://github.com/STEllAR-GROUP/hpx/issues/238
4062 https://github.com/STEllAR-GROUP/hpx/issues/241
4063 https://github.com/STEllAR-GROUP/hpx/issues/247
4064 https://github.com/STEllAR-GROUP/hpx/issues/248
4065 https://github.com/STEllAR-GROUP/hpx/issues/253
4066 https://github.com/STEllAR-GROUP/hpx/issues/272
4067 https://github.com/STEllAR-GROUP/hpx/issues/273
4068 https://github.com/STEllAR-GROUP/hpx/issues/277
4069 https://www.cct.lsu.edu
4070 https://www.lsu.edu
4071 https://stellar-group.org
4072 https://stellar-group.org
4073 https://www.boost.org/
4074 https://stellar-group.org
2.12.2 People

The STE|AR Group (pronounced as stellar) stands for “Systems Technology, Emergent Parallelism, and Algorithm Research”. We are an international group of faculty, researchers, and students working at various institutions around the world. The goal of the STE|AR Group is to promote the development of scalable parallel applications by providing a community for ideas, a framework for collaboration, and a platform for communicating these concepts to the broader community.

Our work is focused on building technologies for scalable parallel applications. HPX, our general purpose C++ runtime system for parallel and distributed applications, is no exception. We use HPX for a broad range of scientific applications, helping scientists and developers to write code which scales better and shows better performance compared to more conventional programming models such as MPI.

HPX is based on ParalleX which is a new (and still experimental) parallel execution model aiming to overcome the limitations imposed by the current hardware and the techniques we use to write applications today. Our group focuses on two types of applications - those requiring excellent strong scaling, allowing for a dramatic reduction of execution time for fixed workloads and those needing highest level of sustained performance through massive parallelism. These applications are presently unable (through conventional practices) to effectively exploit a relatively small number of cores in a multi-core system. By extension, these applications will not be able to exploit high-end exascale computing systems which are likely to employ hundreds of millions of such cores by the end of this decade.

Critical bottlenecks to the effective use of new generation high performance computing (HPC) systems include:

- **Starvation**: due to lack of usable application parallelism and means of managing it,
- **Overhead**: reduction to permit strong scalability, improve efficiency, and enable dynamic resource management,
- **Latency**: from remote access across system or to local memories,
- **Contention**: due to multicores chip I/O pins, memory banks, and system interconnects.

The ParalleX model has been devised to address these challenges by enabling a new computing dynamic through the application of message-driven computation in a global address space context with lightweight synchronization. The work on HPX is centered around implementing the concepts as defined by the ParalleX model. HPX is currently targeted at conventional machines, such as classical Linux based Beowulf clusters and SMP nodes.

We fully understand that the success of HPX (and ParalleX) is very much the result of the work of many people. To see a list of who is contributing see our tables below.

---

4075 https://stellar-group.org
4076 https://stellar-group.org
### HPX contributors

Table 2.39: Contributors

<table>
<thead>
<tr>
<th>Name</th>
<th>Institution</th>
<th>Email</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hartmut Kaiser</td>
<td>Center for Computation and Technology (CCT) (^4077), Louisiana State University (LSU) (^4078)</td>
<td><a href="mailto:hkaiser@cct.lsu.edu">hkaiser@cct.lsu.edu</a></td>
</tr>
<tr>
<td>Thomas Heller</td>
<td>Department of Computer Science 3 - Computer Architecture (^4079), Friedrich-Alexander University Erlangen-Nuremberg (FAU) (^4080)</td>
<td><a href="mailto:thom.heller@gmail.com">thom.heller@gmail.com</a></td>
</tr>
<tr>
<td>Agustin Berge</td>
<td>Center for Computation and Technology (CCT) (^4081), Louisiana State University (LSU) (^4082)</td>
<td><a href="mailto:kaballo86@hotmail.com">kaballo86@hotmail.com</a></td>
</tr>
<tr>
<td>Mikael Simberg</td>
<td>Swiss National Supercomputing Centre (^4083)</td>
<td><a href="mailto:simbergm@cscs.ch">simbergm@cscs.ch</a></td>
</tr>
<tr>
<td>John Biddiscombe</td>
<td>Swiss National Supercomputing Centre (^4084)</td>
<td><a href="mailto:biddisco@cscs.ch">biddisco@cscs.ch</a></td>
</tr>
<tr>
<td>Anton Bikineev</td>
<td>Center for Computation and Technology (CCT) (^4085), Louisiana State University (LSU) (^4086)</td>
<td><a href="mailto:ant.bikineev@gmail.com">ant.bikineev@gmail.com</a></td>
</tr>
<tr>
<td>Martin Stumpf</td>
<td>Department of Computer Science 3 - Computer Architecture (^4087), Friedrich-Alexander University Erlangen-Nuremberg (FAU) (^4088)</td>
<td><a href="mailto:martin.h.stumpf@gmail.com">martin.h.stumpf@gmail.com</a></td>
</tr>
<tr>
<td>Bryce Adelstein Lelbach</td>
<td>NVIDIA (^4089)</td>
<td><a href="mailto:brycelelbach@gmail.com">brycelelbach@gmail.com</a></td>
</tr>
<tr>
<td>Shuangyang Yang</td>
<td>Center for Computation and Technology (CCT) (^4090), Louisiana State University (LSU) (^4091)</td>
<td><a href="mailto:syang16@cct.lsu.edu">syang16@cct.lsu.edu</a></td>
</tr>
<tr>
<td>Jeroen Habraken</td>
<td>Technische Universiteit Eindhoven (^4092)</td>
<td><a href="mailto:jhabraken@cct.lsu.edu">jhabraken@cct.lsu.edu</a></td>
</tr>
<tr>
<td>Steven Brandt</td>
<td>Center for Computation and Technology (CCT) (^4093), Louisiana State University (LSU) (^4094)</td>
<td><a href="mailto:sbrandt@cct.lsu.edu">sbrandt@cct.lsu.edu</a></td>
</tr>
<tr>
<td>Antoine Tran Tan</td>
<td>Center for Computation and Technology (CCT) (^4095), Louisiana State University (LSU) (^4096)</td>
<td><a href="mailto:antoine.trantan@iri.fr">antoine.trantan@iri.fr</a></td>
</tr>
<tr>
<td>Adrian Serio</td>
<td>Center for Computation and Technology (CCT) (^4097), Louisiana State University (LSU) (^4098)</td>
<td><a href="mailto:aserio@cct.lsu.edu">aserio@cct.lsu.edu</a></td>
</tr>
<tr>
<td>Maciej Brodowicz</td>
<td>Center for Research in Extreme Scale Technologies (CREST) (^4099), Indiana University (IU) (^4100)</td>
<td><a href="mailto:mbrodowi@indiana.edu">mbrodowi@indiana.edu</a></td>
</tr>
</tbody>
</table>
### Contributors to this document

<table>
<thead>
<tr>
<th>Name</th>
<th>Institution</th>
<th>Email</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hartmut Kaiser</td>
<td>Center for Computation and Technology (CCT), Louisiana State University (LSU)</td>
<td><a href="mailto:h.kaiser@cct.lsu.edu">h.kaiser@cct.lsu.edu</a></td>
</tr>
<tr>
<td>Thomas Heller</td>
<td>Department of Computer Science 3 - Computer Architecture, Friedrich-Alexander University Erlangen-Nuremberg (FAU)</td>
<td><a href="mailto:thom.heller@gmail.com">thom.heller@gmail.com</a></td>
</tr>
<tr>
<td>Bryce Adelstein Lebach</td>
<td>NVIDIA</td>
<td><a href="mailto:brycelelbach@gmail.com">brycelelbach@gmail.com</a></td>
</tr>
<tr>
<td>Vinay C Amatya</td>
<td>Center for Computation and Technology (CCT), Louisiana State University (LSU)</td>
<td><a href="mailto:vamatya@cct.lsu.edu">vamatya@cct.lsu.edu</a></td>
</tr>
<tr>
<td>Steven Brandt</td>
<td>Center for Computation and Technology (CCT), Louisiana State University (LSU)</td>
<td><a href="mailto:sbrandt@cct.lsu.edu">sbrandt@cct.lsu.edu</a></td>
</tr>
<tr>
<td>Maciej Brodowicz</td>
<td>Center for Research in Extreme Scale Technologies (CREST), Indiana University (IU)</td>
<td><a href="mailto:mbrodowi@indiana.edu">mbrodowi@indiana.edu</a></td>
</tr>
<tr>
<td>Adrian Serio</td>
<td>Center for Computation and Technology (CCT), Louisiana State University (LSU)</td>
<td><a href="mailto:aserio@cct.lsu.edu">aserio@cct.lsu.edu</a></td>
</tr>
</tbody>
</table>

4077 https://www.cct.lsu.edu  
4078 https://www.lsu.edu  
4079 https://www3.cs.fau.de  
4080 https://www.fau.de  
4081 https://www.cct.lsu.edu  
4082 https://www.lsu.edu  
4083 https://www.cscs.ch  
4084 https://www.cscs.ch  
4085 https://www.cct.lsu.edu  
4086 https://www.lsu.edu  
4087 https://www3.cs.fau.de  
4088 https://www.fau.de  
4089 https://nvidia.com/  
4090 https://www.cct.lsu.edu  
4091 https://www.lsu.edu  
4092 https://www.iui.nl  
4093 https://www.cct.lsu.edu  
4094 https://www.lsu.edu  
4095 https://www.cct.lsu.edu  
4096 https://www.lsu.edu  
4097 https://www.cct.lsu.edu  
4098 https://www.lsu.edu  
4099 https://pti.iu.edu  
4100 https://www.iu.edu  
4101 https://www.cct.lsu.edu  
4102 https://www.lsu.edu  
4103 https://www3.cs.fau.de  
4104 https://www.fau.de  
4105 https://nvidia.com/  
4106 https://www.cct.lsu.edu  
4107 https://www.lsu.edu  
4108 https://www.cct.lsu.edu  
4109 https://www.lsu.edu  
4110 https://pti.iu.edu  
4111 https://www.iu.edu  
4112 https://www.cct.lsu.edu  
4113 https://www.lsu.edu

Chapter 2. What’s so special about **HPX**?
Acknowledgements

Thanks also to the following people who contributed directly or indirectly to the project through discussions, pull requests, documentation patches, etc.

- Auriane Reverdell (Swiss National Supercomputing Centre\footnote{https://www.cscs.ch}), for her tireless work on refactoring our CMake setup and modularizing HPX.
- Christopher Hinz, for his work on refactoring our CMake setup.
- Weile Wei, for fixing HPX builds with CUDA on Summit.
- Severin Strobl, for fixing our CMake setup related to linking and adding new entry points to the HPX runtime.
- Rebecca Stobaugh, for her major documentation review and contributions during and after the 2019 Google Season of Documentation.
- Jan Melech, for adding automatic serialization of simple structs.
- Austin McCartney, for adding concept emulation of the Ranges TS bidirectional and random access iterator concepts.
- Marco Diers, reporting and fixing issues related PMIx.
- Maximilian Bremer, for reporting multiple issues and extending the component migration tests.
- Piotr Mikolajczyk, for his improvements and fixes to the set and count algorithms.
- Grant Rostig, for reporting several deficiencies on our web pages.
- Jakub Golinsonski, for implementing an HPX backend for OpenCV and in the process improving documentation and reporting issues.
- Mikael Simberg (Swiss National Supercomputing Centre\footnote{https://www.cscs.ch}), for his tireless help cleaning up and maintaining HPX.
- Tianyi Zhang, for his work on HPXMP.
- Shahrzad Shirzad, for her contributions related to Phylanx.
- Christopher Ogle, for his contributions to the parallel algorithms.
- Surya Priy, for his work with statistic performance counters.
- Anushi Maheshwari, for her work on random number generation.
- Bruno Pitrus, for his work with parallel algorithms.
- Nikunj Gupta, for rewriting the implementation of \texttt{hpx\_main\_hpp} and for his fixes for tests.
- Christopher Taylor, for his interest in HPX and the fixes he provided.
- Shoshana Jakobovits, for her work on the resource partitioner.
- Denis Blank, who re-wrote our unwrapped function to accept plain values arbitrary containers, and properly deal with nested futures.
- Ajai V. George, who implemented several of the parallel algorithms.
- Taeguk Kwon, who worked on implementing parallel algorithms as well as adapting the parallel algorithms to the Ranges TS.
- Zach Byerly (Louisiana State University (LSU)\footnote{https://www.lsu.edu}), who in his work developing applications on top of HPX opened tickets and contributed to the HPX examples.
• Daniel Estermann, for his work porting HPX to the Raspberry Pi.
• Alireza Kheirkhahan (Louisiana State University (LSU))\(^{4117}\), who built and administered our local cluster as well as his work in distributed IO.
• Abhimanyu Rawat, who worked on stack overflow detection.
• David Pfander, who improved signal handling in HPX, provided his optimization expertise, and worked on incorporating the Vc vectorization into HPX.
• Denis Demidov, who contributed his insights with VexCL.
• Khalid Hasanov, who contributed changes which allowed to run HPX on 64Bit power-pc architectures.
• Zahra Khatami (Louisiana State University (LSU))\(^{4118}\), who contributed the prefetching iterators and the persistent auto chunking executor parameters implementation.
• Marcin Copik, who worked on implementing GPU support for C++AMP and HCC. He also worked on implementing a HCC backend for HPX.Compute.
• Minh-Khanh Do, who contributed the implementation of several segmented algorithms.
• Bibek Wagle (Louisiana State University (LSU))\(^{4119}\), who worked on fixing and analyzing the performance of the parcel coalescing plugin in HPX.
• Lukas Troska, who reported several problems and contributed various test cases allowing to reproduce the corresponding issues.
• Andreas Schaefer, who worked on integrating his library (LibGeoDecomp\(^{4120}\)) with HPX. He reported various problems and submitted several patches to fix issues allowing for a better integration with LibGeoDecomp\(^{4121}\).
• Satyaki Upadhyay, who contributed several examples to HPX.
• Brandon Cordes, who contributed several improvements to the inspect tool.
• Harris Brakmic, who contributed an extensive build system description for building HPX with Visual Studio.
• Parsa Amini (Louisiana State University (LSU))\(^{4122}\), who refactored and simplified the implementation of AGAS in HPX and who works on its implementation and optimization.
• Luis Martinez de Bartolome who implemented a build system extension for HPX integrating it with the Conan\(^{4123}\) C/C++ package manager.
• Vinay C Amatya (Louisiana State University (LSU))\(^{4124}\), who contributed to the documentation and provided some of the HPX examples.
• Kevin Huck and Nick Chaimov (University of Oregon)\(^{4125}\), who contributed the integration of APEX (Autonomic Performance Environment for eXascale) with HPX.
• Francisco Jose Tapia, who helped with implementing the parallel sort algorithm for HPX.
• Patrick Diehl, who worked on implementing CUDA support for our companion library targeting GPGPUs (HPXCL\(^{4126}\)).
• Eric Lemanissier contributed fixes to allow compilation using the MingW toolchain.

\(^{4117}\) https://www.lsu.edu
\(^{4118}\) https://www.lsu.edu
\(^{4119}\) https://www.lsu.edu
\(^{4120}\) https://www.libgeodecomp.org/
\(^{4121}\) https://www.libgeodecomp.org/
\(^{4122}\) https://www.lsu.edu
\(^{4123}\) https://www.conan.io/
\(^{4124}\) https://www.lsu.edu
\(^{4125}\) https://uooregon.edu/
\(^{4126}\) https://github.com/STEllAR-GROUP/hpxcl/
• Nidhi Makhijani who helped cleaning up some enum consistencies in HPX and contributed to the resource manager used in the thread scheduling subsystem. She also worked on HPX in the context of the Google Summer of Code 2015.

• Larry Xiao, Devang Bacharwar, Marcin Copik, and Konstantin Kronfeldner who worked on HPX in the context of the Google Summer of Code program 2015.

• Daniel Bourgeois (Center for Computation and Technology (CCT)\textsuperscript{4127}) who contributed to HPX the implementation of several parallel algorithms (as proposed by N4313\textsuperscript{4128}).

• Anuj Sharma and Christopher Bross (Department of Computer Science 3 - Computer Architecture\textsuperscript{4129}), who worked on HPX in the context of the Google Summer of Code\textsuperscript{4130} program 2014.

• Martin Stumpf (Department of Computer Science 3 - Computer Architecture\textsuperscript{4131}), who rebuilt our contiguous testing infrastructure (see the HPX Buildbot Website\textsuperscript{4132}). Martin is also working on HPXCL\textsuperscript{4133} (mainly all work related to OpenCL\textsuperscript{4134}) and implementing an HPX backend for POCL\textsuperscript{4135}, a portable computing language solution based on OpenCL\textsuperscript{4136}.

• Grant Mercer (University of Nevada, Las Vegas\textsuperscript{4137}), who helped creating many of the parallel algorithms (as proposed by N4313\textsuperscript{4138}).

• Damond Howard (Louisiana State University (LSU)\textsuperscript{4139}), who works on HPXCL\textsuperscript{4140} (mainly all work related to CUDA\textsuperscript{4141}).

• Christoph Junghans (Los Alamos National Lab), who helped making our buildsystem more portable.

• Antoine Tran Tan (Laboratoire de Recherche en Informatique, Paris), who worked on integrating HPX as a backend for NT2\textsuperscript{4142}. He also contributed an implementation of an API similar to Fortran co-arrays on top of HPX.

• John Biddiscombe (Swiss National Supercomputing Centre\textsuperscript{4143}), who helped with the BlueGene/Q port of HPX, implemented the parallel sort algorithm, and made several other contributions.

• Erik Schnetter (Perimeter Institute for Theoretical Physics), who greatly helped to make HPX more robust by submitting a large amount of problem reports, feature requests, and made several direct contributions.

• Mathias Gaunard (Metascale), who contributed several patches to reduce compile time warnings generated while compiling HPX.

• Andreas Buhr, who helped with improving our documentation, especially by suggesting some fixes for inconsistencies.

• Patricia Grubel (New Mexico State University\textsuperscript{4144}), who contributed the description of the different HPX thread scheduler policies and is working on the performance analysis of our thread scheduling subsystem.

\textsuperscript{4127} https://www.cct.lsu.edu
\textsuperscript{4128} http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2014/n4313.html
\textsuperscript{4129} https://www3.cs.fau.de
\textsuperscript{4130} https://developers.google.com/open-source/soc/
\textsuperscript{4131} https://www3.cs.fau.de
\textsuperscript{4132} http://rostam.cct.lsu.edu/
\textsuperscript{4133} https://github.com/STEllAR-GROUP/hpxcl/
\textsuperscript{4134} https://www.khronos.org/opencl/
\textsuperscript{4135} https://portablecl.org/
\textsuperscript{4136} https://www.khronos.org/opencl/
\textsuperscript{4137} https://www.unlv.edu
\textsuperscript{4138} http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2014/n4313.html
\textsuperscript{4139} https://www.lsu.edu
\textsuperscript{4140} https://github.com/STEllAR-GROUP/hpxcl/
\textsuperscript{4141} https://www.nvidia.com/object/cuda_home_new.html
\textsuperscript{4142} https://www.numscale.com/nt2/
\textsuperscript{4143} https://www.cs.cscs.ch
\textsuperscript{4144} https://www.nmsu.edu
• Lars Viklund, whose wit, passion for testing, and love of odd architectures has been an amazing contribution to our team. He has also contributed platform specific patches for FreeBSD and MSVC12.
• Agustin Berge, who contributed patches fixing some very nasty hidden template meta-programming issues. He rewrote large parts of the API elements ensuring strict conformance with C++11/14.
• Anton Bikineev for contributing changes to make using boost::lexical_cast safer, he also contributed a thread safety fix to the iostreams module. He also contributed a complete rewrite of the serialization infrastructure replacing Boost.Serialization inside HPX.
• Pyry Jahkola, who contributed the Mac OS build system and build documentation on how to build HPX using Clang and libc++.
• Mario Mulansky, who created an HPX backend for his Boost.Odeint library, and who submitted several test cases allowing us to reproduce and fix problems in HPX.
• Rekha Raj, who contributed changes to the description of the Windows build instructions.
• Jeremy Kemp how worked on an HPX OpenMP backend and added regression tests.
• Alex Nagelberg for his work on implementing a C wrapper API for HPX.
• Chen Guo, helvihartmann, Nicholas Pezolano, and John West who added and improved examples in HPX.
• Joseph Kleinhenz, Markus Elfring, Kirill Kropivynansky, Alexander Neundorf, Bryant Lam, and Alex Hirsch who improved our CMake.
• Tapasweni Pathak, Praveen Velliengiri, Jean-Loup Tastet, Michael Levine, Aalekh Nigam, HadrienG2, Prayag Verma, Islada, Alex Myczko, and Ayyav Kumar who improved the documentation.
• Jayesh Badwaik, J. F. Bastien, Christoph Garth, Christopher Hinz, Brandon Kohn, Mario Lang, Maikel Nadolski, pierrele, hendrx, Dekken, woodmeister123, xaguilar, Andrew Kemp, Dylan Stark, Matthew Anderson, Jeremy Wilke, Jiazheng Yuan, CyberDrudge, david8dixon, Maxwell Reeser, Raffaele Solca, Marco Ippolito, Jules Penuchot, Weile Wei, Severin Strobl, Kor de Jong, albestro, Jeff Trull, and Yuri Victorovich who contributed to the general improvement of HPX.

In addition to the people who worked directly with HPX development we would like to acknowledge the NSF, DoE, DARPA, Center for Computation and Technology (CCT)4145, Department of Computer Science 3 - Computer Architecture4146, and Swiss National Supercomputing Centre4147 who fund and support our work. We would also like to thank the following organizations for granting us allocations of their compute resources: LSU HPC, LONI, XSEDE, NERSC, and the Gauss Center for Supercomputing.

HPX is currently funded by the following grants:
• The National Science Foundation through awards 1240655 (STAR), 1339782 (STORM), and 1737785 (Phyllanx). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.
• The Department of Energy (DoE) through the awards DE-AC52-06NA25396 (FLeCSI) and DE-NA0003525 (Resilience). Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

4145 https://www.cct.lsu.edu
4146 https://www3.cs.fau.de
4147 https://www.cscs.ch
• The Defense Technical Information Center (DTIC) under contract FA8075-14-D-0002/0007. Neither the United States Government nor any agency thereof, nor any of their employees makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights.

• The Bavarian Research Foundation (Bayerische Forschungsstiftung) through the grant AZ-987-11.

• The European Commission’s Horizon 2020 programme through the grant H2020-EU.1.2.2. 671603 (AllScale).
CHAPTER 3

Index

• genindex