HPX – The Futurization of Computing

Thomas Heller (thomas.heller@cs.fau.de)
February 28, 2014
Department Computer Science 3
Friedrich-Alexander-Universität Erlangen-Nürnberg
The HPX Programming Model

Towards a C++ compliant Interface and beyond

Case Studies
   HPX Thread Granularity
   LibGeoDecomp
   ZERPA
   ... and more
The HPX Programming Model
The 4 Horsemen of the Apocalypse: SLOW

Starvation
Latency
Overhead
Waiting for contention
Fighting SLOWness: Governing Principles

- *Active global address space (AGAS)*, instead of PGAS
- *Message driven*, instead of Message Passing
- *Lightweight Control Objects*, instead of Global Barriers
- *Adaptive locality control*, instead of Static Data Distribution
- *Moving work to data*, instead of Moving Data to Work
- *Fine grained parallelism of lightweight threads*, instead of Communication Sequential Processes (CSP/MPI)
HPX – A general purpose runtime system

A uniform, standards-oriented API for ease of programming parallel and distributed applications.

⇒ Standard C++ codebase
⇒ Fully C++11/14 compliant
HPX – A general purpose runtime system

*Exposing new and unexpected forms of parallelism.*

⇒ Enables programmer to write fully asynchronous code using hundreds of millions of threads.
⇒ Provides unified syntax and semantics for local and remote operations.
⇒ Makes concurrency manageable with dataflow and future based synchronization.
HPX – A general purpose runtime system

*Implements a rich set of runtime services supporting a broad range of use cases.*

⇒ Introspect the state of your parallel computer at any time
⇒ Performance Counters
⇒ Debugger?
HPX – A general purpose runtime system

*Has been designed and developed for systems of any scale.*

⇒ Currently running on ARM, x86, Xeon Phi, BlueGeneQ
⇒ Supporting Windows, Linux, OSX, Android, CNK
⇒ Ranging from large scale Clusters over Desktop Computers to Handheld devices
⇒ *Existing performant implementation*
Is the first fully functional implementation of the ParalleX execution model.

⇒ ParalleX is the theoretic foundation on which HPX was built
HPX – A general purpose runtime system

Is published under a liberal open-source license and has a open, active, and thriving developer community.

⇒ Boost License Version 1.0
⇒ Over 30 contributors from all over the world
⇒ Development started 2005
Towards a C++ compliant Interface and beyond
Why C++?

- Widely accepted industry standard
- C is a subset of C++!
- Powerful mechanisms to build abstractions while maintaining high performance
Parallelism in C++

- Threads (thread)
- Futures (future<T>, shared_future<T>)
- Asynchronous Tasks (async)
- Synchronization primitives (mutex, condition_variable)
- Atomics
- Memory Model
Utilities

- Partial function application (`bind`)
- Heterogeneous tuples (`tuple<T...>`)  
- Generic (type erased) function objects (`function<R(...)>>`)
## HPX Extensions: Actions

<table>
<thead>
<tr>
<th>R f(p...)</th>
<th>Synchronous (returns R)</th>
<th>Asynchronous (returns future&lt;R&gt;)</th>
<th>Fire &amp; Forget (returns void)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functions (direct)</td>
<td>f(p...)</td>
<td>async(f, p...)</td>
<td>apply(f, p...)</td>
</tr>
<tr>
<td>Functions (lazy)</td>
<td>bind(f, p...)(...)</td>
<td>async(bind(f, p...), ...)</td>
<td>apply(bind(f, p...), ...)</td>
</tr>
<tr>
<td>Actions (direct)</td>
<td>HPX_ACTION(f, a) a()(id, p...)</td>
<td>HPX_ACTION(f, a) async(a(), id, p...)</td>
<td>HPX_ACTION(f, a) apply(a(), id, p...)</td>
</tr>
<tr>
<td>Actions (lazy)</td>
<td>HPX_ACTION(f, a) bind(a(), id, p...) (…)</td>
<td>HPX_ACTION(f, a) async(bind(a(), id, p...), ...)</td>
<td>HPX_ACTION(f, a) apply(bind(a(), id, p...), ...)</td>
</tr>
</tbody>
</table>
HPX Extensions: Components

- C++ Objects in the Global Address space
- Referencable through unique GIDs
- Referencable through meaningful symbolic names
- Member functions callable through actions
- Migratable
HPX Extensions: More power to futures

Composable futures
- `hpx::when_all`, `hpx::when_any`, `hpx::when_n`
- `hpx::future<T>::then`
- `hpx::dataflow`

Expressing locality
- Executors let you specify where your tasks run and how they are scheduled
The HPX Programming Model

Key features

- Fully asynchronous
  - ’Fire & Forget’ semantics (no result available)
  - ’Pure’ asynchronous semantics (result available through `hpx::future`)
  - Can be used ’synchronously’, but does not block

- Fully type safe remote operations
  - Extending the notion of a callable to remote case (action)
  - Everything you can do with functions, can be done with actions

- Data types can be used in remote contexts
  - Can be sent over the wire (`hpx::function`, `hpx::bind`, `hpx::any`)
  - Can be used with actions (`hpx::bind`, `hpx::async`, `hpx::function`)
Case Studies
Homogeneous thread spawning in HPX (weak scaling)

Courtesy of Bryce of Lelbach (LSU) and Patricia Grubel (NMSU)
LibGeoDecomp

- C++ Auto-parallelizing framework
- Open Source
- High scalability
- Wide range of platform support
- http://www.libgeodecomp.org
LibGeoDecomp
Futurizing the Simulation Flow

Basic Simulation flow:

```java
for (Region r: innerRegion) {
    update(r, oldGrid, newGrid, step);
}
swap(oldGrid, newGrid);
++step;
for (Region r: outerGhostZoneRegion) {
    notifyPatchProviders(r, oldGrid);
}
for (Region r: outerGhostZoneRegion) {
    update(r, oldGrid, newGrid, step);
}
for (Region r: innerGhostZoneRegion) {
    notifyPatchAccepters(r, oldGrid);
}
```
LibGeoDecomp
Futurizing the Simulation Flow

Futurized Simulation flow:

```c
parallel for (Region r: innerRegion) {
    update (r, oldGrid, newGrid, step);
}
```

Continuation

```c
swap (oldGrid, newGrid); ++step;
parallel for (Region r: outerGhostZoneRegion) {
    notifyPatchProviders (r, oldGrid);
}
```

Continuation

```c
parallel for (Region r: outerGhostZoneRegion) {
    update (r, oldGrid, newGrid, step);
}
```

Continuation

```c
parallel for (Region r: innerGhostZoneRegion) {
    notifyPatchAccepters (r, oldGrid);
}
```
LibGeoDecomp
Performance Results

Execution Times of HPX and MPI N-Body Codes
(SMP, Weak Scaling)

- Sim HPX
- Sim MPI
- Comm HPX
- Comm MPI

Number of Cores, on one Node
LibGeoDecomp
Performance Results
LibGeoDecomp
Performance Results

Weak Scaling Results for HPX N-Body Code
(Single Xeon Phi, Futurized)
LibGeoDecomp

Performance Results

Weak Scaling Results for HPX N-Body Codes
(Host Cores and Xeon Phi Accelerator)

Performance in TFLOPS

Number of Nodes, 16 Cores on Host, Full Xeon Phi
ZERPA: Taking futurization to the next level

- An EDSL for describing interdependant operators on Volumes
- Heavy usage of the HPX dataflow programming capabilities
ZERPA: Taking futurization to the next level

cast_part = loader("cast_part")
, mask = loader("mask")
, filtered = cross_median(cast_part)
, cleaned = special_median(filtered, mask)
ZERPA: Taking futurization to the next level

mask = loader

Gussteil = loader

filtered = cross_median<1,1,1>

cleaned = special_median<9,9,9>

Requesting result

Pushing completed result
ZERPA: Taking futurization to the next level

One block per core

Optimal block size

CPU time

loader  cross_median  special_median
HPX OpenCL integration

- Seamless integration of calling OpenCL kernels inside of HPX
- An HPX backend for pocl
- https://github.com/STEllAR-GROUP/hpxcl
Other recent results

Tiled LU-Decomposition

![Graph showing execution time for Tiled LU-Decomposition. The graph compares HPX, Dataflow, and Plasma.](image)

Courtesy of Antoine Tran Tan (LRI, Université Paris-Sud XI, INRIA - Orsay, France)
Other recent results

Migration path: OpenMP Backend implemented on top of HPX

Courtesy of Jeremy A. Kemp (UH)
Other recent results

OpenMP vs. HPX
Barriers vs. Dataflow

```c
for(int i = 0; i < num_iterations; ++i)
{
    // compute something depending on i-1, i and j
}
```

```c
std::vector<future<T>> results[];
for(int i = 0; i < num_iterations; ++i)
{
    for(j = 0; j < N; ++j)
    {
        results[i][j] = when_all(dependencies).then(
            // compute something depending on i-1, i and j
        );
    }
}
```
Other recent results

OpenMP vs. HPX: Jacobi Smoother
Barriers vs. Dataflow
- github.com/STEllAR-GROUP/hpx/wiki/GSoC-2014-Project-Ideas
- www.google-melange.com/gsoc/homepage/google/gsoc2014
Summary: Revolution through Evolution

- HPX is a general purpose runtime system
- Standards oriented API
- Unified local and remote semantics
- Existing performant implementation
Get in touch!

- **Blog**: [http://stellar.cct.lsu.edu](http://stellar.cct.lsu.edu)
- **Code**: [https://github.com/STEllAR-GROUP/hpx](https://github.com/STEllAR-GROUP/hpx)
- **Mailing List**: hpx-users@stellar.cct.lsu.edu
- **IRC**: #stellar @ irc.freenode.org