HPX
High Performance ParalleX
CCT – Tech Talk Series

Hartmut Kaiser (hkaiser@cct.lsu.edu)
http://stellar.cct.lsu.edu
What’s HPX?

- Exemplar ParalleX runtime system implementation
  - Targeting conventional architectures (Linux based SMPs and clusters)
  - Currently, mainly software only implementation
  - Emphasis on
    - Functionality: finding proper architecture and API’s
    - Performance: finding hotspots, contention points, reduce overheads, hide latencies
    - API: finding the minimal but complete set of required functions
    - Driven by real applications (AMR, Contact, Graphs, CFD)

- Should allow retargeting to different platforms
  - Stable basis for long term migration path of applications
  - Highly modular, allows to incorporate different policy implementations
  - First experiments with custom hardware components

- Implemented in C++
  - Utilize compiler for highly optimized implementation
  - Utilize language for best possible user experience/simplest API
Technology Demands new Response

Figure courtesy of Kunle Olukotun, Lance Hammond, Herb Sutter, and Burton Smith

http://stellar.cct.lsu.edu
Technology Demands new Response

Figure courtesy of Kunle Olukotun, Lance Hammond, Herb Sutter, and Burton Smith

http://stellar.cct.lsu.edu
Amdahl’s Law (Strong Scaling)

\[ S = \frac{1}{(1 - P) + \frac{P}{N}} \]

- \( S \): Speedup
- \( P \): Proportion of parallel code
- \( N \): Number of processors

Figure courtesy of Wikipedia (http://en.wikipedia.org/wiki/Amdahl’s_law)
Gustafson’s Law (Weak Scaling)

\[ S = N - P(N - 1) \]

- **S**: Speedup
- **P**: Proportion of parallel code
- **N**: Number of processors
The 4 Horsemen of the Apocalypse: **SLOW**

- **Starvation**
  - Insufficient concurrent work to maintain high utilization of resources

- **Latencies**
  - Time-distance delay of remote resource access and services

- **Overheads**
  - Work for management of parallel actions and resources on critical path which are not necessary in sequential variant

- **Waiting for Contention resolution**
  - Delays due to lack of availability of oversubscribed shared resource
The 4 Horsemen of the Apocalypse: SLOW

- **Starvation**
  - Insufficient concurrent work to maintain high utilization of resources

- **Latencies**
  - Time-distance delay of remote access and services

- **Overheads**
  - Work for management actions and which are on critical path which are not necessary in sequential variant

- **Waiting for Contention**
  - Delays due to lack of availability of oversubscribed shared resource

Impose upper bound on both, weak and strong scaling
Main HPX Runtime System Tasks

- Manage parallel execution for the application
  - Exposing parallelism, runtime adaptive management of parallelism and resources
  - Synchronizing parallel tasks
  - Thread (task) scheduling, load balancing
- Mitigate latencies for the application
  - Latency hiding through overlap of computation and communication
  - Latency avoidance through locality management
- Reduce overhead for the application
  - Synchronization, scheduling, load balancing, communication, context switching, memory management, address translation
- Resolve contention for the application
  - Adaptive routing, resource scheduling, load balancing
  - Localized request buffering for logical resources

Starvation
Latency
Overhead
Contention
What’s ParalleX?

- Active global address space (AGAS) instead of PGAS
- Message driven instead of message passing
- Lightweight control objects instead of global barriers
- Latency hiding instead of latency avoidance
- 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 Communicating Sequential Processes (CSP/MPI)
HPX Runtime System Design

- Current version of HPX provides the following infrastructure on conventional systems as defined by the ParalleX execution model
  - Active Global Address Space (AGAS)
  - ParalleX Threads and ParalleX Thread Management
  - Parcel Transport and Parcel Management
  - Local Control Objects (LCOs)
  - ParalleX Processes
    - Namespace and policies management, locality control
    - Monitoring subsystem
HPX Runtime System Design

- Current version of HPX provides the following infrastructure on conventional systems as defined by the ParalleX execution model:
  - Active Global Address Space (AGAS)
  - ParalleX Threads and ParalleX Thread Management
  - Parcel Transport and Parcel Management
  - Local Control Objects (LCOs)
  - ParalleX Processes
  - Namespace and policies management, locality control
  - Monitoring subsystem
Active Global Address Space

- Global Address Space throughout the system
  - Removes dependency on static data distribution
  - Enables dynamic load balancing of application and system data
- AGAS assigns global names (identifiers, unstructured 128 bit integers) to all entities managed by HPX.
- Unlike PGAS provides a mechanism to resolve global identifiers into corresponding local virtual addresses (LVA)
  - LVAs comprise – Locality ID, Type of Entity being referred to and its local memory address
  - Moving an entity to a different locality updates this mapping
  - Current implementation is based on centralized database storing the mappings which are accessible over the local area network.
  - Local caching policies have been implemented to prevent bottlenecks and minimize the number of required round-trips.
- Current implementation allows autonomous creation of globally unique ids in the locality where the entity is initially located and supports memory pooling of similar objects to minimize overhead
- Implemented garbage collection scheme of HPX objects
Active Global Address Space

- Global Address Space throughout the system
  - Removes dependency on static data distribution
  - Enables dynamic load balancing of application and system data
- AGAS assigns global names (identifiers, unstructured 128 bit integers) to all entities managed by HPX.
- Unlike PGAS provides a mechanism to resolve global identifiers into corresponding local virtual addresses (LVAs).
  - LVAs comprise Locality ID, Type of Entity being referred to and its local memory address.
  - Moving an entity updates this mapping.
  - Current implementation is based on a centralized database storing the mappings which are accessible over the local area network.
  - Local caching policies have been implemented to prevent bottlenecks and minimize the number of required round-trips.
- Current implementation allows autonomous creation of globally unique ids in the locality where the entity is initially located and supports memory pooling of similar objects to minimize overhead.
- Implemented garbage collection scheme of HPX objects.
Parcel Management

- Active messages (parcels)
  - Destination address, function to execute, parameters, continuation
- Any inter-locality messaging is based on Parcels
  - In HPX parcels are represented as polymorphic objects
- An HPX entity on creating a parcel object hands it to the parcel handler.
- The parcel handler serializes the parcel where all dependent data is bundled along with the parcel
- At the receiving locality the parcel is de-serialized and causes a HPX thread to be created based on its content
Parcel Management

- Active messages (parcels)
  - Destination address, function to execute, parameters, continuation
- Any inter-locality messaging is based on Parcels
  - In HPX parcels are represented as polymorphic objects
- An HPX entity on creating a parcel object hands it to the parcel handler.
- The parcel handler serializes the parcel where all dependent data is bundled along with the parcel
- At the receiving locality the parcel is de-serialized and causes an HPX thread to be created based on its content
Thread Management

- Thread manager is modular and implements a work-queue based management as specified by the ParalleX execution model
- Threads are cooperatively scheduled at user level without requiring a kernel transition
- Specially designed synchronization primitives such as semaphores, mutexes etc. allow synchronization of PX-threads in the same way as conventional threads
- Thread management currently supports several key modes
  - Global Thread Queue
  - Local Queue (work stealing)
  - Local Priority Queue (work stealing)
Thread Management

- Thread manager is modular and implements a work-queue based management as specified by the ParalleX execution model.
- Threads are cooperatively scheduled at user level without requiring a kernel transition.
- Specially designed synchronization primitives such as semaphores, mutexes, etc. allow synchronization of PX threads in the same way as conventional threads.
- Thread management currently supports several key modes:
  - Global Thread Queue
  - Local Queue (work stealing)
  - Local Priority Queue (work stealing)

http://stellar.cct.lsu.edu
Constraint based Synchronization

- Compute dependencies at task instantiation time
- No global barriers, uses constraint based synchronization
- Computation flows at its own pace
- Message driven
- Symmetry between local and remote task creation/execution
- Possibility to control grain size
LCOs (Local Control Objects)

- LCOs provide a means of controlling parallelization and synchronization of PX-threads
- Enable event-driven thread creation and can support in-place data structure protection and on-the-fly scheduling
- Preferably embedded in the data structures they protect
- Abstraction of a multitude of different functionalities for
  ▫ event driven PX-thread creation,
  ▫ protection of data structures from race conditions
  ▫ automatic on-the-fly scheduling of work
- LCO may create (or reactivate) a PX-thread as a result of ‘being triggered’
LCOs (Local Control Objects)

- LCOs provide a means of controlling parallelization and synchronization of PX-threads
- Enable event-driven thread creation and on-the-fly scheduling
- Preferably embedded in the data structures they protect
- Abstract a multitude of different functionalities:
  - event-driven PX-thread creation
  - protection of data structures from race conditions
  - automatic on-the-fly scheduling of work
- LCO may create (or reactivate) a PX-thread as a result of 'being triggered'

Diagram:

- Process Manager
- Local Memory Management
- Performance Monitor
- Interconnect
- AGAS Address Translation
- Action Manager
- Parcel Port
- Parcel Handler
- Thread Manager
- Thread Pool
- LCOs

http://stellar.cct.lsu.edu
Exemplar LCO: Futures

• In HPX Futures LCO refers to an object that acts as a proxy for the result that is initially not known.

• When a user code invokes a future (using future.get() ) the thread can do one of 2 activities
  ▫ If the remote data/arguments are available then the future.get() operation fetches the data and the execution of the thread continues
  ▫ If the remote data is NOT available the thread may continue until it requires the actual value; then the thread suspends allowing other threads to continue execution. The original thread re-activates as soon as the data dependency is resolved
ParalleX Processes

- Management of namespace and locality
  - Centerpiece for truly distributed AGAS
  - We completed the first step in re-implementing AGAS towards being distributed

- Encapsulation of blocks of functionality and possibly distributed data
  - Completed software architecture design for processes
  - Implemented prototypical processes managing read-only distributed data
Recent Results

- First formal release of HPX3 (V0.6), V1.0 scheduled for May
- Re-implemented AGAS on top of Parcel transport layer
  - Better distributed scalability, better asynchrony, latency hiding
- First results implementing ParalleX processes
  - Distributed (read-only) data partitioning of very large data sets
- First (encouraging) results from distributed runs
  - Demonstrated strong scaling similar to SMP
- Consolidated performance counter monitoring framework
  - Allows to measure almost arbitrary system characteristics using unified interface
- Implemented remote application steering
  - Used to demonstrate control of power consumption
- New applications
  - Linear algebra, N-body, chess, contact, graph500
Scaling & performance: AMR using MPI and HPX

![Scaling of MPI AMR application](http://stellar.cct.lsu.edu)
Scaling & performance: AMR using MPI and HPX

Scaling of MPI AMR application

Levels of AMR refinement

Scaling of HPX AMR application

Levels of AMR refinement

1 core
2 cores
4 cores
10 cores
20 cores
Scaling & performance: AMR using MPI and HPX

Wallclock time ratio MPI/HPX
(Depending on levels of refinement - LoR, pollux.cct.lsu.edu, 32 cores)

- 1 core
- 2 cores
- 5 cores
- 10 cores
- 20 cores
- 30 cores

Levels of AMR refinement:
- 0 LoR
- 1 LoR
- 2 LoR
- 3 LoR

Scaling (normalized to 1 core)

0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14

http://stellar.cct.lsu.edu
Distributed Strong Scaling
Distributed Strong Scaling

Wallclock Time

Strong Scaling for Dataflow

http://stellar.cct.lsu.edu
Overheads: AGAS

AGAS Overhead Benchmark
(10000 Futures)

Walltime (Normalized to 1 OS-Thread)

AGAS Worker OS-Threads

- No caching
- Caching
- Ranged caching
Overhead: Threads

![Graph showing execution time vs. number of OS threads (cores)](http://stellar.cct.lsu.edu)

- 0μs
- 3.5μs
- 7μs
- 14.5μs
- 29μs
- 58μs
- 115μs

Execution Time [s] (1,000,000 PX Threads)

http://stellar.cct.lsu.edu
Overhead: Threads

Execution Time [s] for 100,000 Futures

- Execution Time for 100,000 Futures with different overhead times:
  - 0 μs
  - 10 μs
  - 19 μs
  - 38 μs
  - 76 μs
  - 150 μs
  - 300 μs

Number of OS threads (cores) vs. Execution Time [s]
Results from a Gauss-Seidel Solver

![Graph showing execution time vs. grain size for different numbers of cores.](http://stellar.cct.lsu.edu)
Results from a Gauss-Seidel Solver

**Execution Time**
(Depending on Grain Size)

**Strong Scaling for Gauss-Seidel Solver**
(Matrix size: 2000x2000, Grain size: 90x90)

**Scaling (normalized to 1 core)**
Grain Size: The New Freedom
Overhead: Load Balancing

Competing effects for optimal grain size: overheads vs. load balancing (starvation)
Overhead: Load Balancing

Competing effects for optimal grain size: overheads vs. load balancing (starvation)
Conclusions

• Are we there yet?
  ▫ Definitely NO!
  ▫ But very promising results supporting our claim

• Are we on a right path?
  ▫ Definitely YES!
  ▫ Might not be THE right path, but it’s a leap

• Do we have cure for those scaling impaired applications?
  ▫ We’re not sure yet!
  ▫ Based on results we are optimistic