ParalleX

A Cure for Scaling Impaired Parallel Applications

Hartmut Kaiser (hkaiser@cct.lsu.edu)

Tianhe-1A 2.566 Petaflops Rmax

Heterogeneous Architecture:

- 14,336 Intel Xeon CPUs
- 7,168 Nvidia Tesla M2050 GPUs
- More than 100 racks
- 4.04 megawatts

Technology Demands new Response

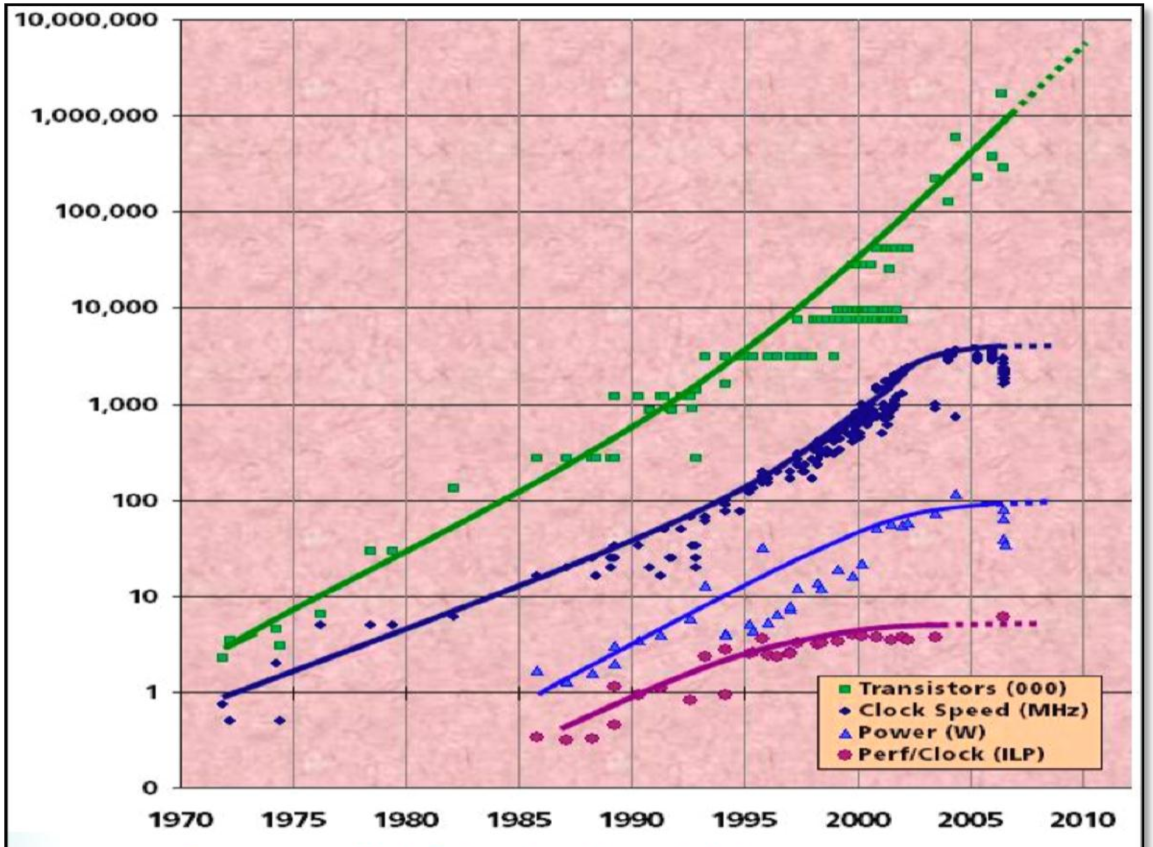


Figure courtesy of Kunle Olukotun, Lance Hammond, Herb Sutter, and Burton Smith

Technology Demands new Response

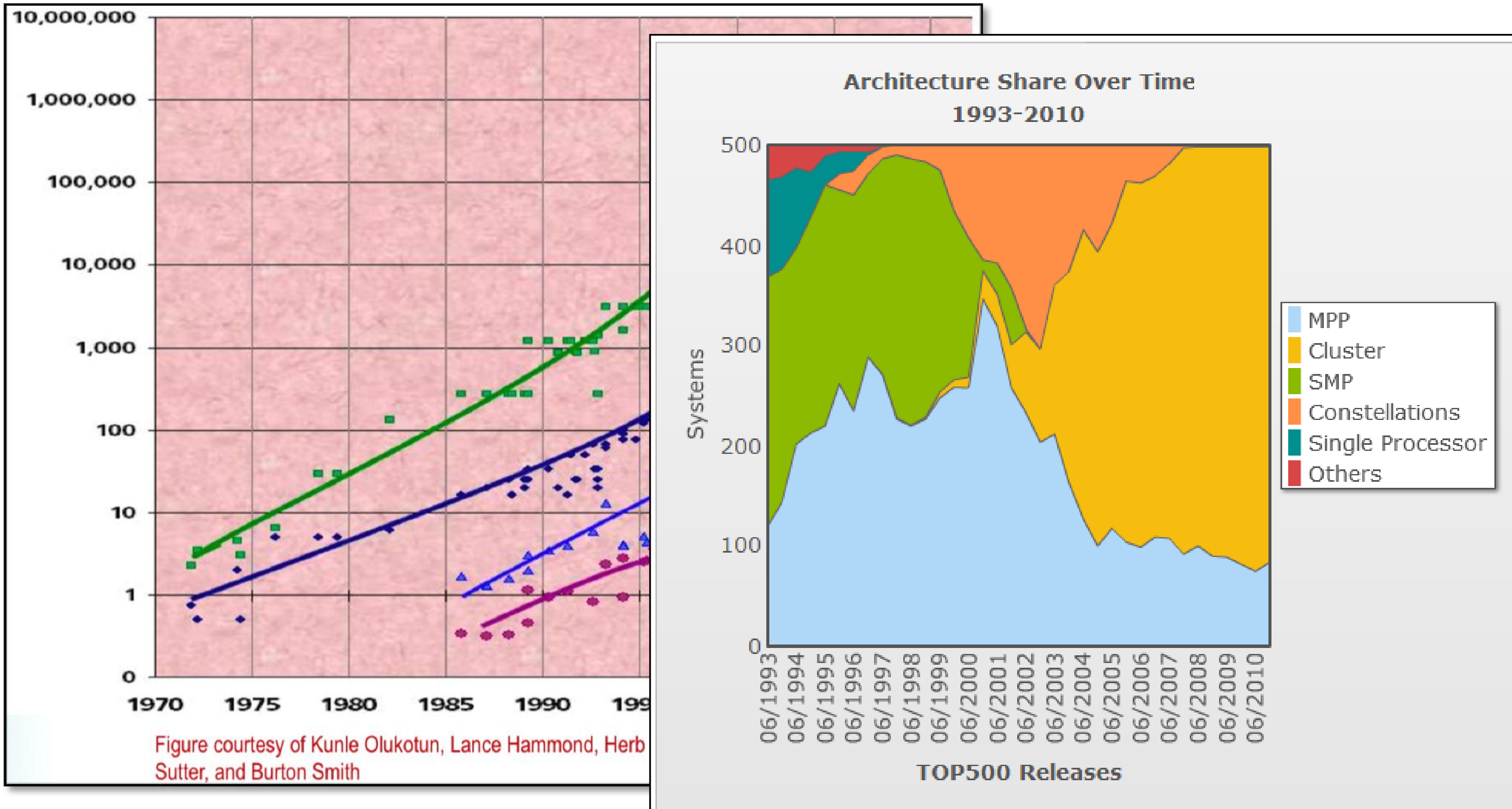


Figure courtesy of Kunle Olukotun, Lance Hammond, Herb Sutter, and Burton Smith

Amdahl's Law

$$\frac{1}{(1-P)+\frac{P}{N}}$$

- P: Proportion of parallel code
- N: Number of processors

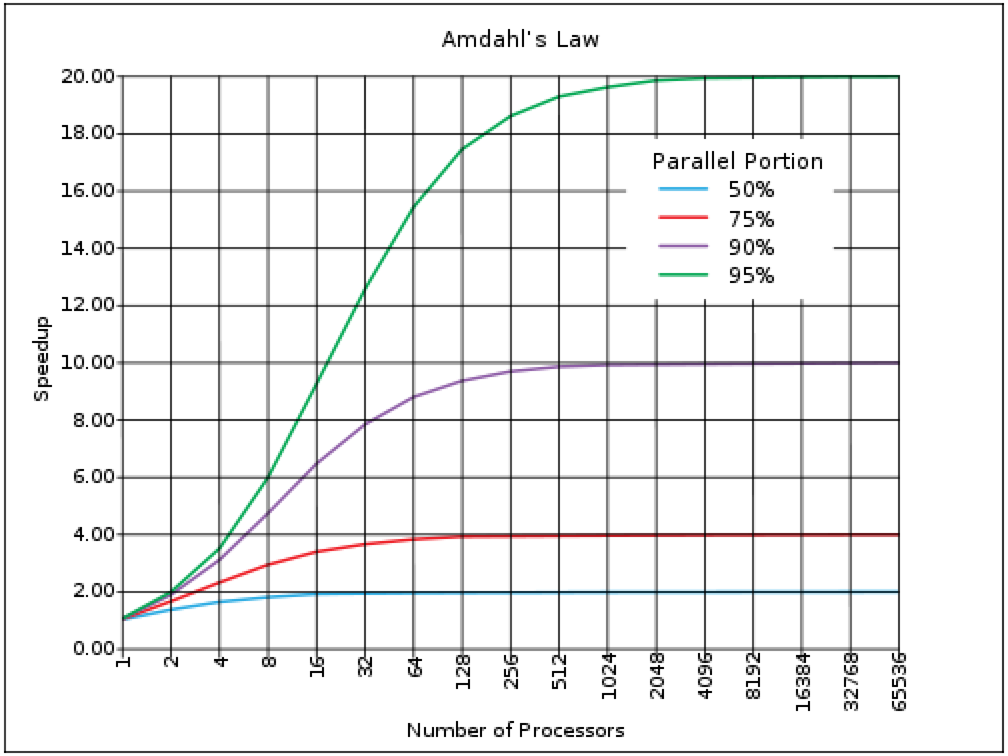


Figure courtesy of Wikipedia (http://en.wikipedia.org/wiki/Amdahl's_law)

The 4 Horsemen of the Apocalypse: SLOW

- **S**tarvation
- **L**atencies
- **O**verheads
- **W**aiting for Contention resolution


courtesy of www.albrecht-durer.org


courtesy of www.albrecht-durer.org

Efficiency Factors

- Starvation
  - Insufficient concurrent work to maintain high utilization of resources
    - Inadequate global or local parallelism due to poor load balancing
- Latency
  - Time-distance delay of remote resource access and services
    - E.g., memory access and system-wide message passing
- Overhead
  - Critical path work for management of parallel actions and resources
  - Work not necessary for sequential variant
- Waiting for contention resolution
  - Delay due to lack of availability of oversubscribed shared resource
    - Bottlenecks in the system, e.g., memory bank access, and network bandwidth

Efficiency Factors

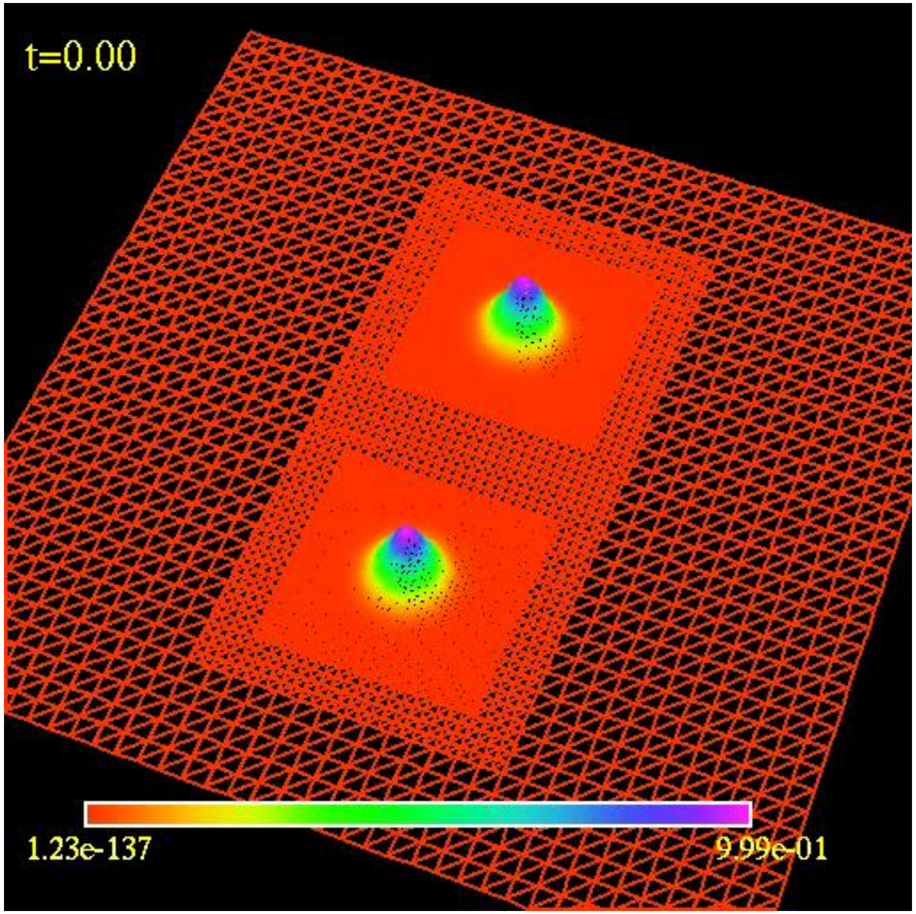
- Starvation
  - Insufficient concurrent work to maintain hig…
    - Inadequate global or local parallelism due to p…
- Latency
  - Time-distance delay of rem… …ces
    - E.g., memory access an…
- Overhead
  - Critical p… …parallel actions and resources
  - W… …al variant
- … …solution
  - … …availability of oversubscribed shared resource
    - … the system, e.g., memory bank access, and network bandwidth

Impose upper bound on both weak and strong scaling

The Runtime System

A Game Changer

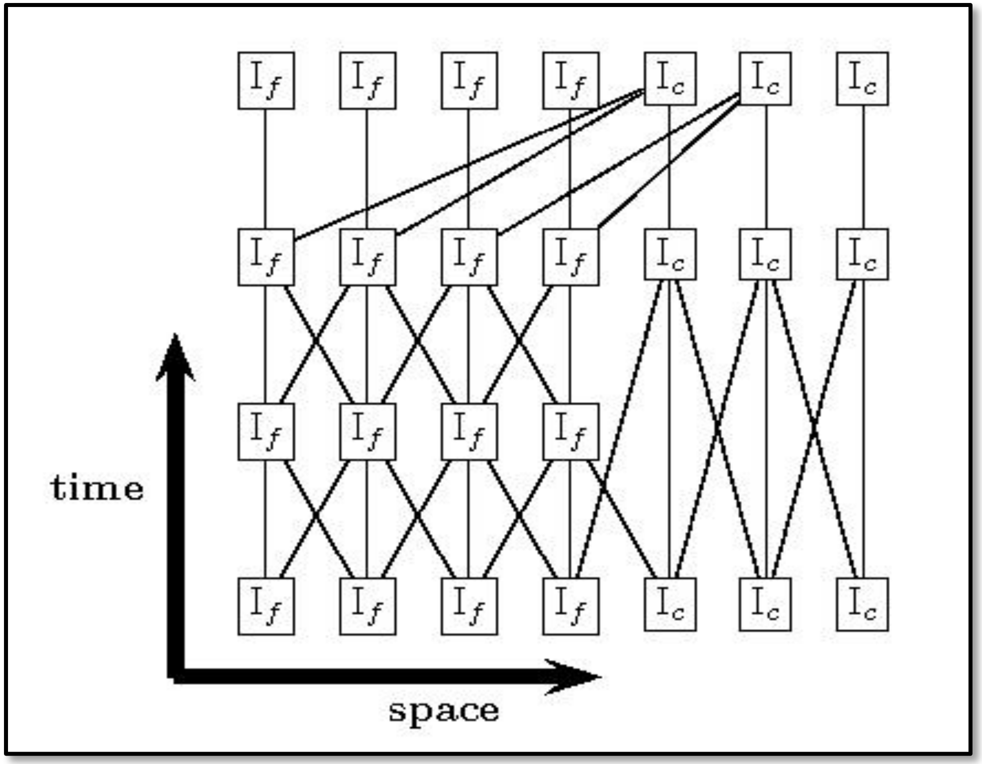
Adaptive Mesh Refinement (AMR)



Why Adaptive Mesh Refinement (AMR)

- From 31 Mar 2010 to 31 Mar 2011 at least 68,394,791 SU's were dedicated on Teragrid to finite difference based AMR applications (out of ~1.407 billion SU's allocated) -- about 5% of runs
- Nearly all of the publicly available AMR toolkits use MPI
- Strong scaling of AMR applications is typically very poor
- ParalleX functionality fits nicely with the AMR algorithm: global address space, "work stealing", parallelism discovery, dynamic threads, implicit load balancing

Constraint based Synchronization for AMR



- 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

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
- Fine grained parallelism of lightweight threads instead of Communicating Sequential Processes (CSP/MPI)
- Moving work to data instead of moving data to work

The Runtime System – A Game Changer

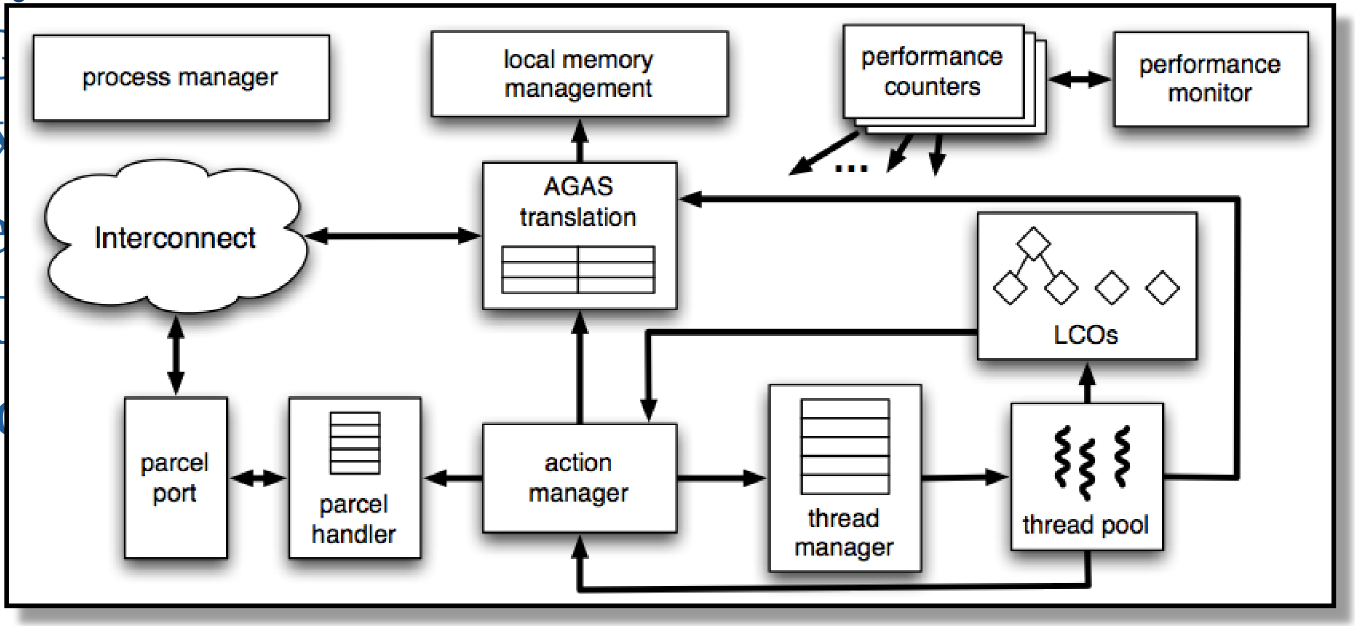
- Runtime system
  - is: ephemeral, dedicated to and exists only with an application
  - is not: the OS, persistent and dedicated to the hardware system
- Moves us from *static* to *dynamic* operational regime
  - Exploits situational awareness for causality-driven adaptation
  - Guided-missile with continuous course correction rather than a fired projectile with fixed-trajectory
- Based on foundational assumption
  - Untapped system resources to be harvested
  - More computational work will yield reduced time and lower power
  - Opportunities for enhanced efficiencies discovered only in flight
  - New methods of control to deliver superior scalability
- "Undiscovered Country" – adding a dimension of systematics
  - Adding a new component to the system stack
  - Path-finding through the new trade-off space

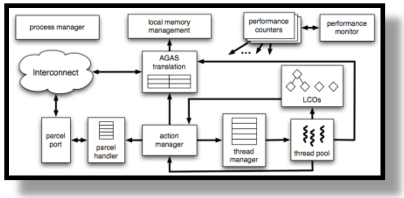
HPX Runtime System Design

- 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)

HPX Runtime System Design

- Current version of HPX provides the following infrastructure on conventional systems as defined by the ParalleX execution model
  - Active G
  - ParalleX Manage
  - Parcel T
  - Local C

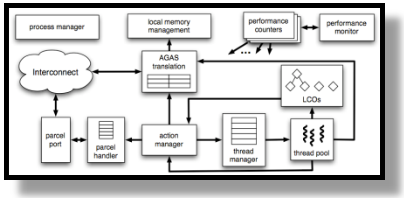




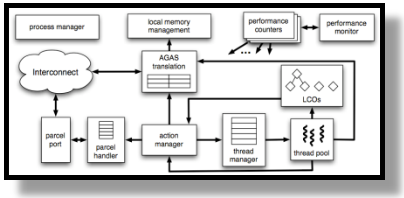
Main Runtime System Tasks

- Manage parallel execution for application — **Starvation**
  - Delineating parallelism, runtime adaptive management of parallelism
  - Synchronizing parallel tasks
  - Thread scheduling, static and dynamic load balancing
- Mitigate latencies for application — **Latencies**
  - Latency hiding through overlap of computation and communication
  - Latency avoidance through locality management
  - Dynamic copy semantic support
- Reduce overhead for application — **Overheads**
  - Synchronization, scheduling, load balancing, communication, context switching, memory management, address translation
- Resolve contention for application — **Contention**
  - Adaptive routing, resource scheduling, load balancing
  - Localized request buffering for logical resources

Active Global Address Space

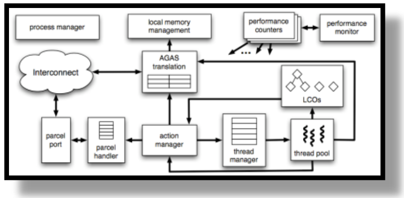


- Global Address Space throughout the system
  - Removes dependency on static data distribution
  - Enables dynamic load balancing of application and system data
- AGAS assigns global names (identifiers, unstructured 128 bit integers to all entities managed by HPX.
- Unlike PGAS allows mechanisms to resolving 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



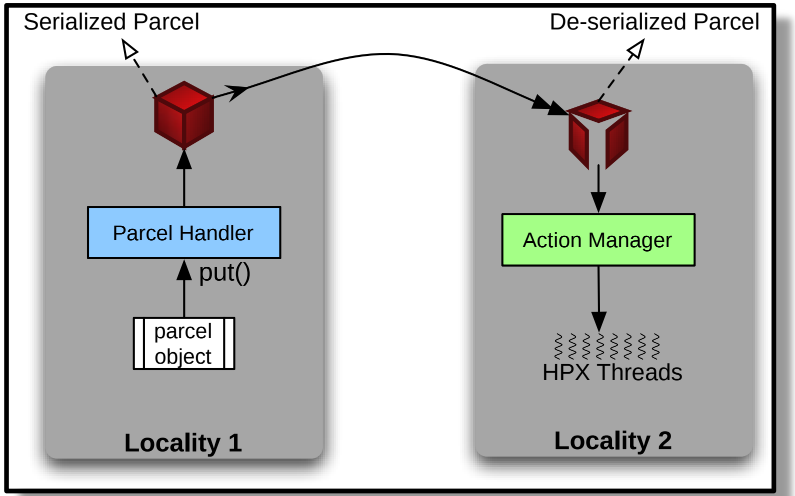
Thread Management

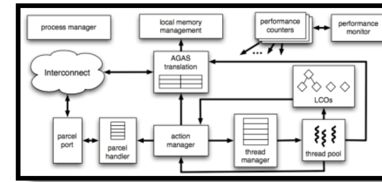
- Thread manager is modular and implements a work-queue based management as specified by PX Execution model
- Threads are cooperatively scheduled at user level without requiring a kernel transition
- Specially designed synchronization primitives such as semaphores, mutexes etc. allow synchronization of HPX 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)



Parcel Management

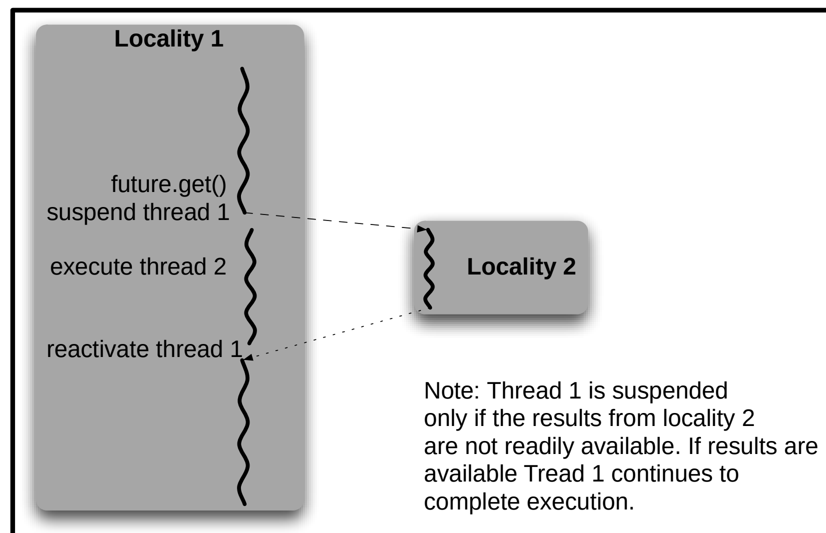
- Any inter-locality messaging is based on Parcels
  - In HPX implementation parcels are represented as polymorphic objects
  - An HPX entity on creating a parcel object sends it to the parcel handler.
- The parcel handler serializes the parcel where all dependent data is bundled along with the parcel.
- At the receiving locality the parcel is received using the standard TCP/IP protocols,
- The action manager de-serializes the parcel and creates HPX threads out of the specification





Exemplar LCO: Futures

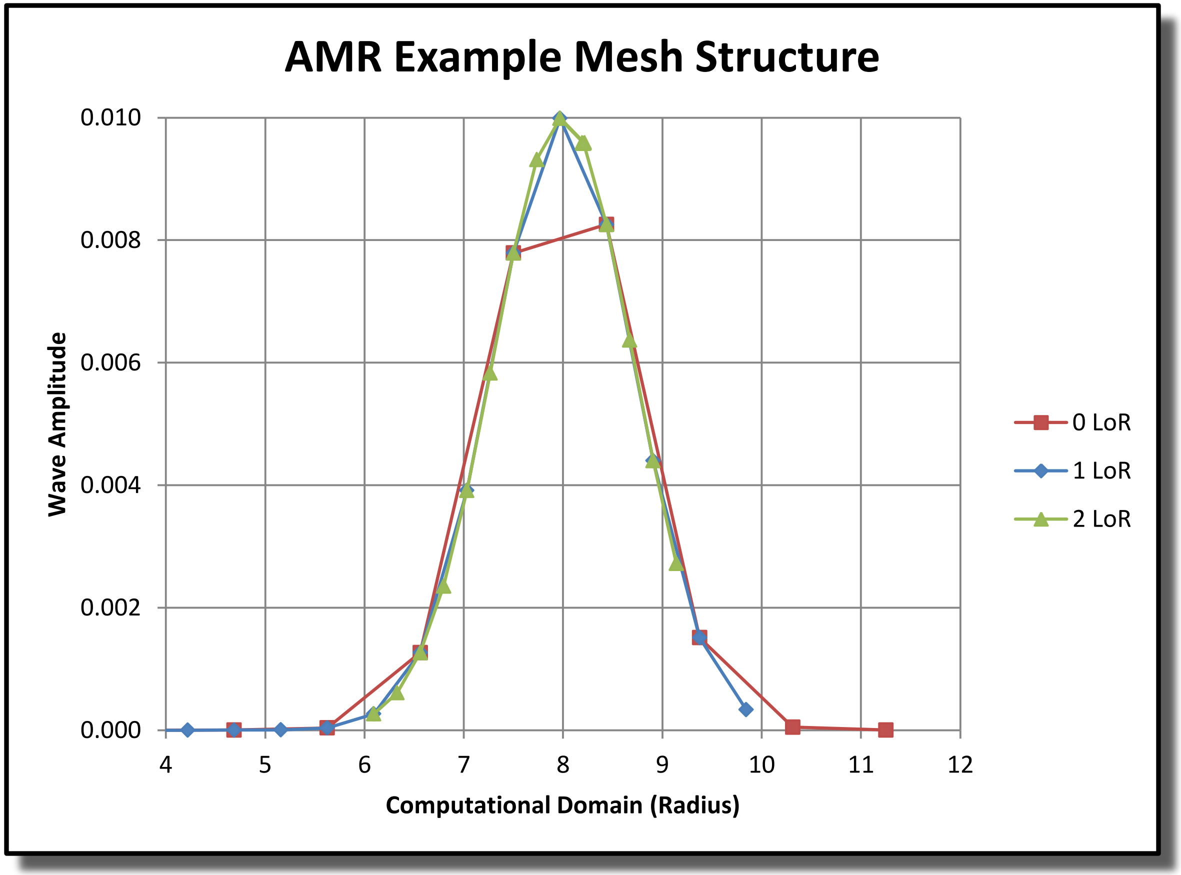
- In HPX Futures LCO refers to an object that acts as a proxy for the result that is initially not known.
- When a user code invokes a future (using future.get()) the thread can do one of 2 activities
  - If the remote data /arguments are available then the future.get() operation fetches the data and 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 data dependency is resolved



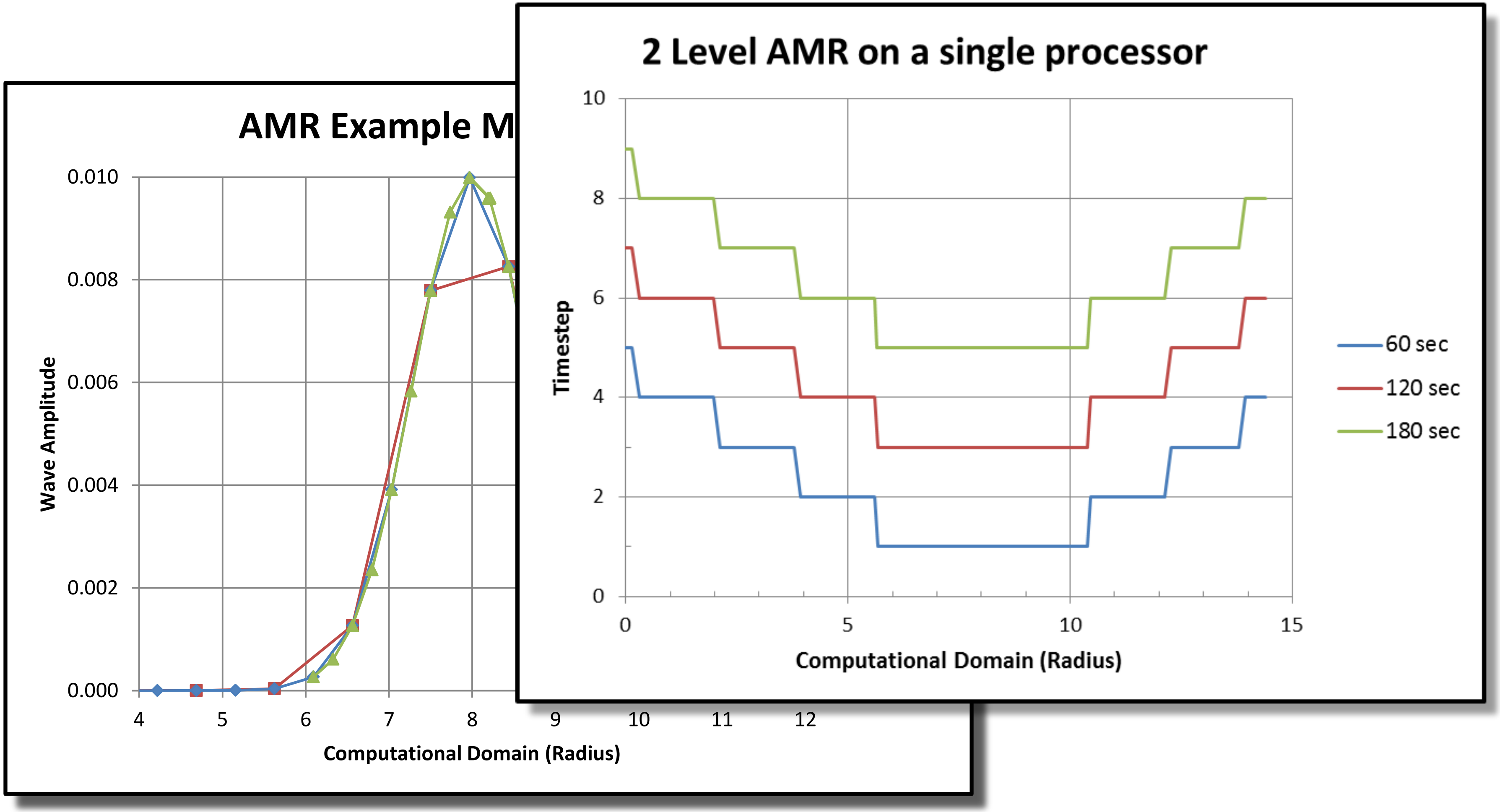
First Results

Based on HPX – An exemplar implementation of ParalleX for conventional systems

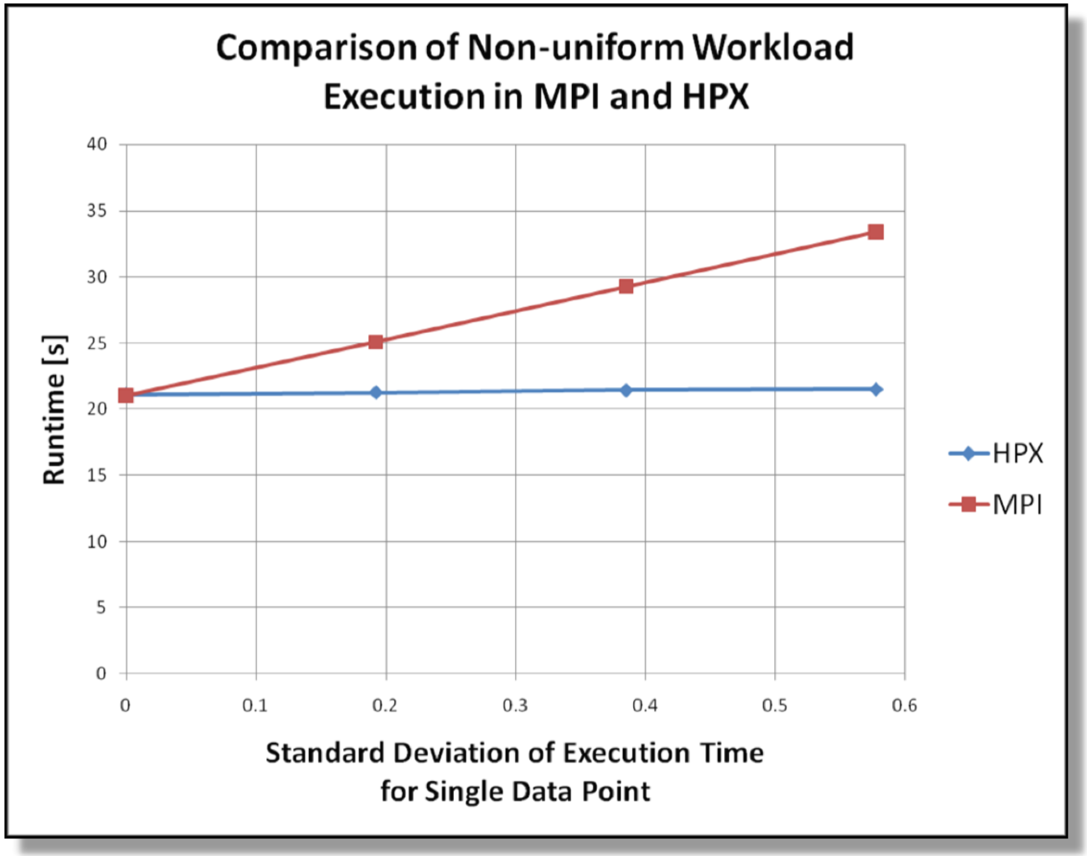
Starvation: Non-uniform Workload



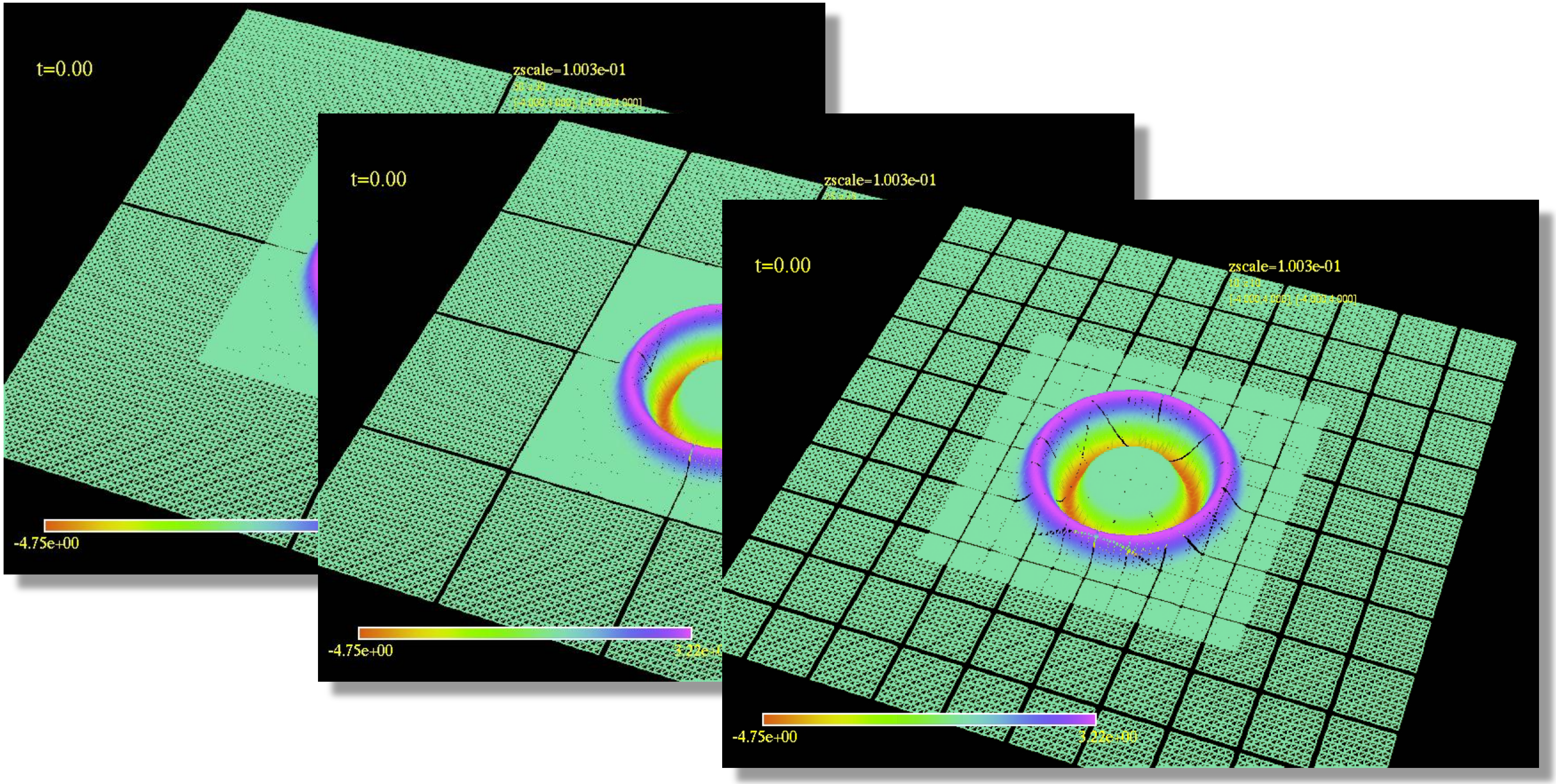
Starvation: Non-uniform Workload



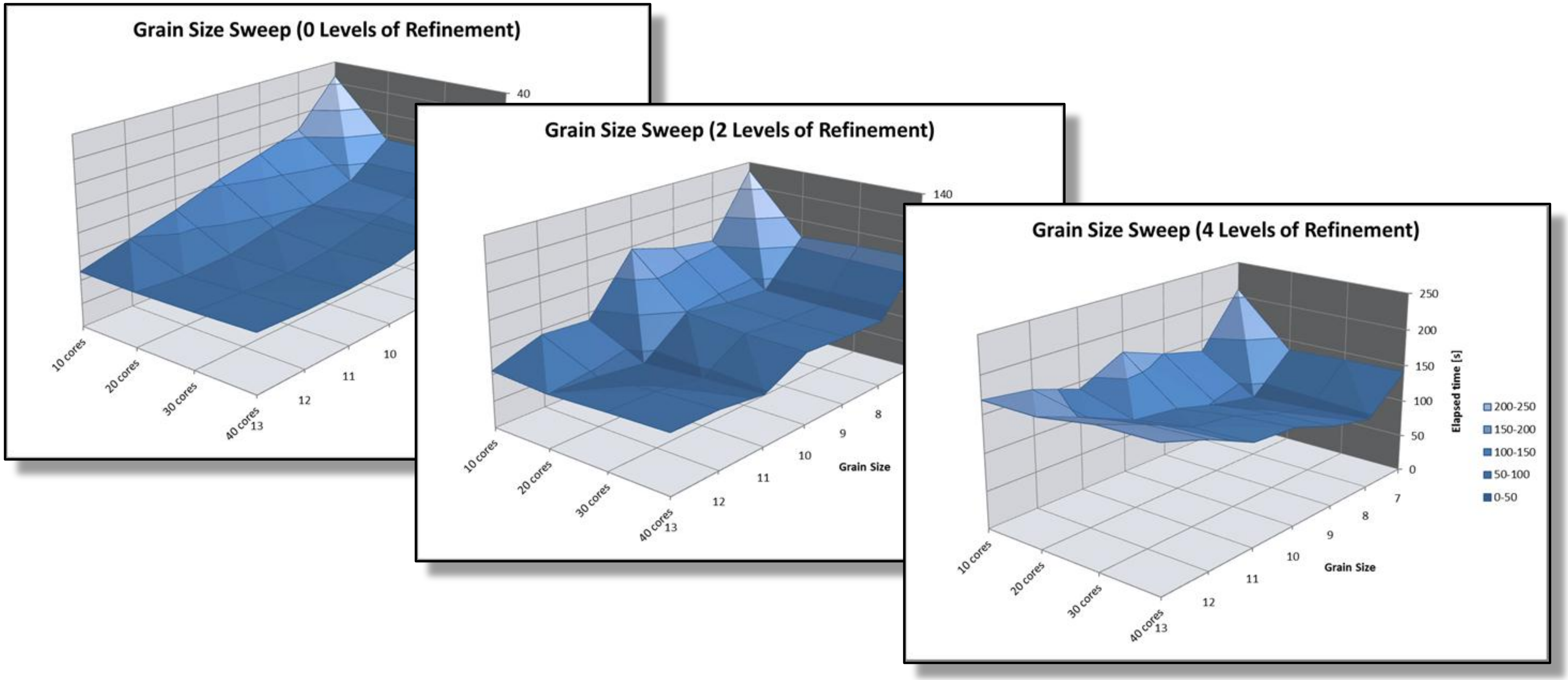
Starvation: Non-uniform Workload



Grain Size: The New Freedom

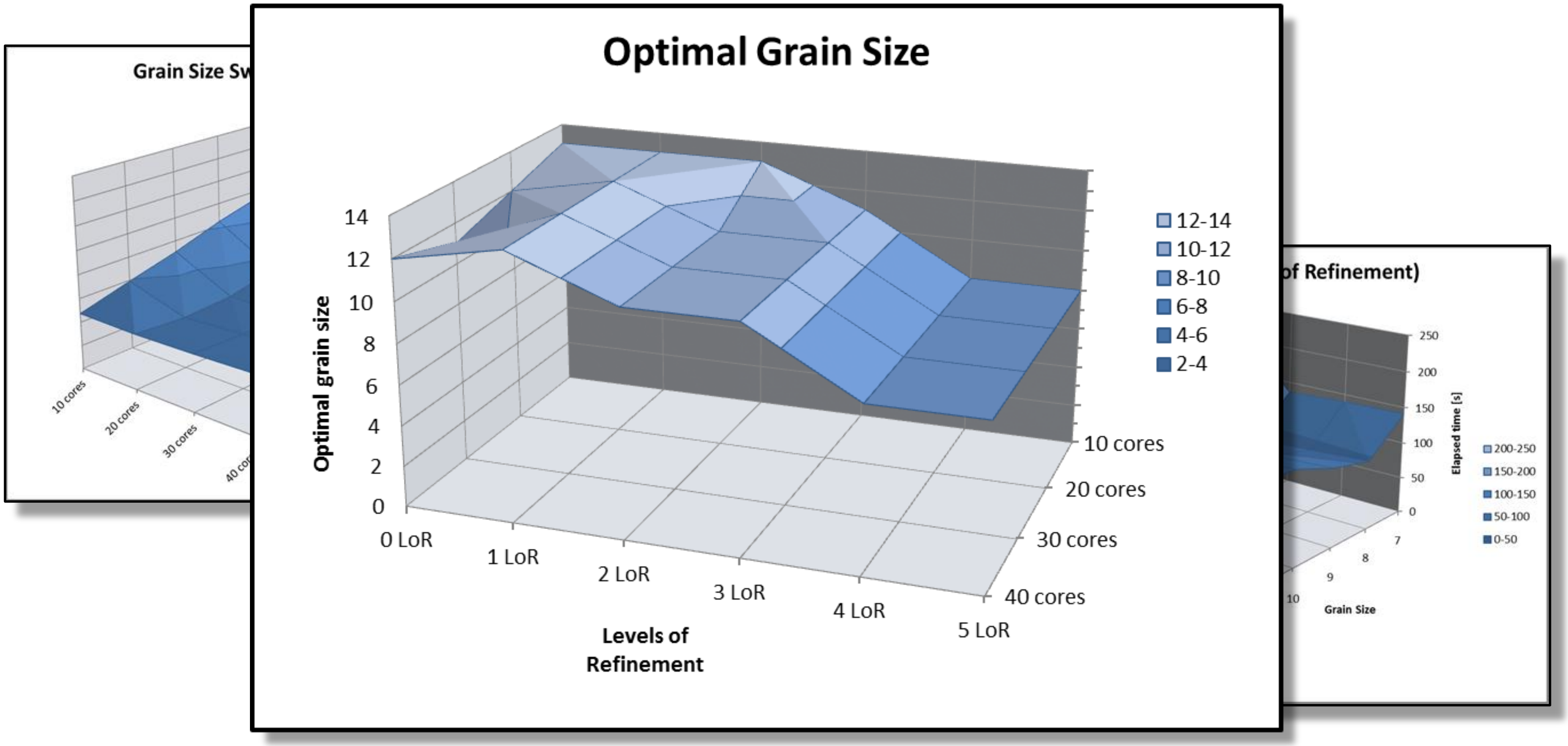


Overhead: Load Balancing



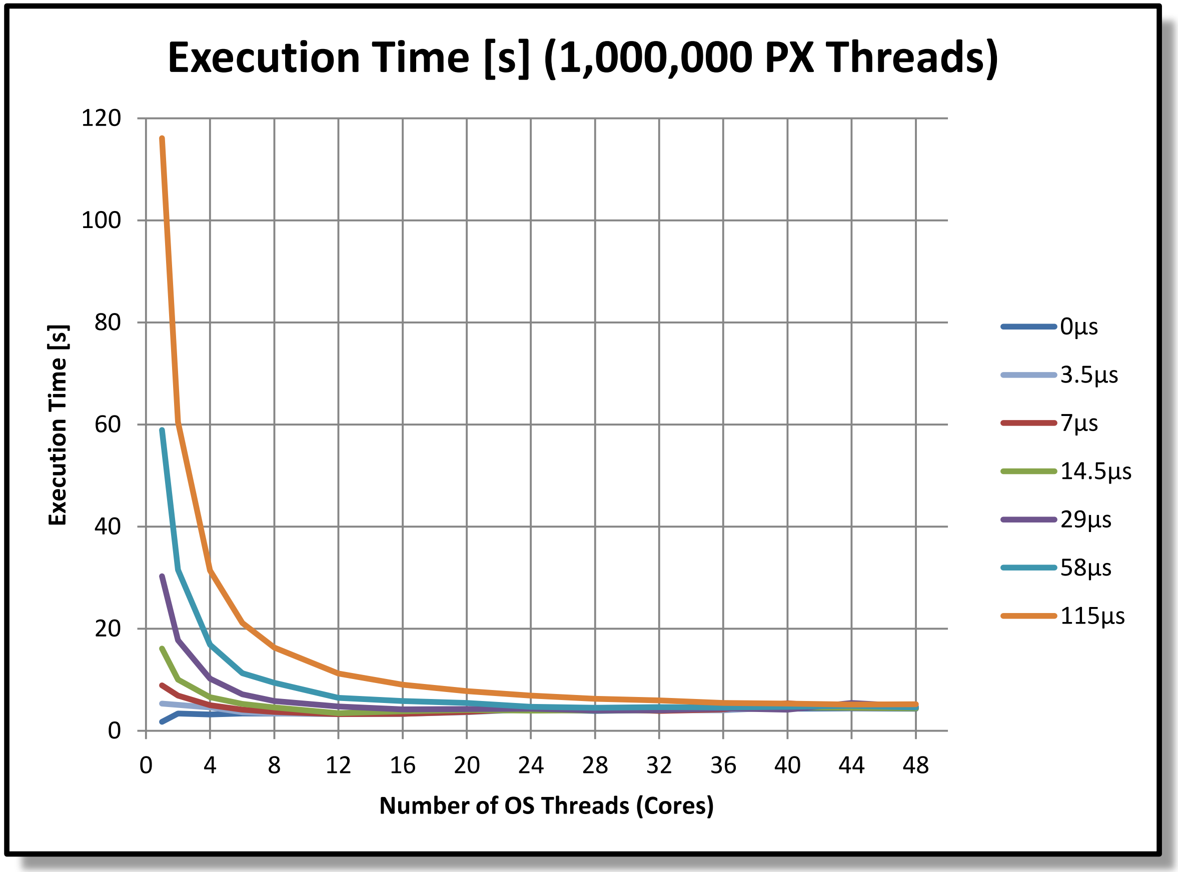
Competing effects for optimal grain size: overheads vs. load balancing (starvation)

Overhead: Load Balancing

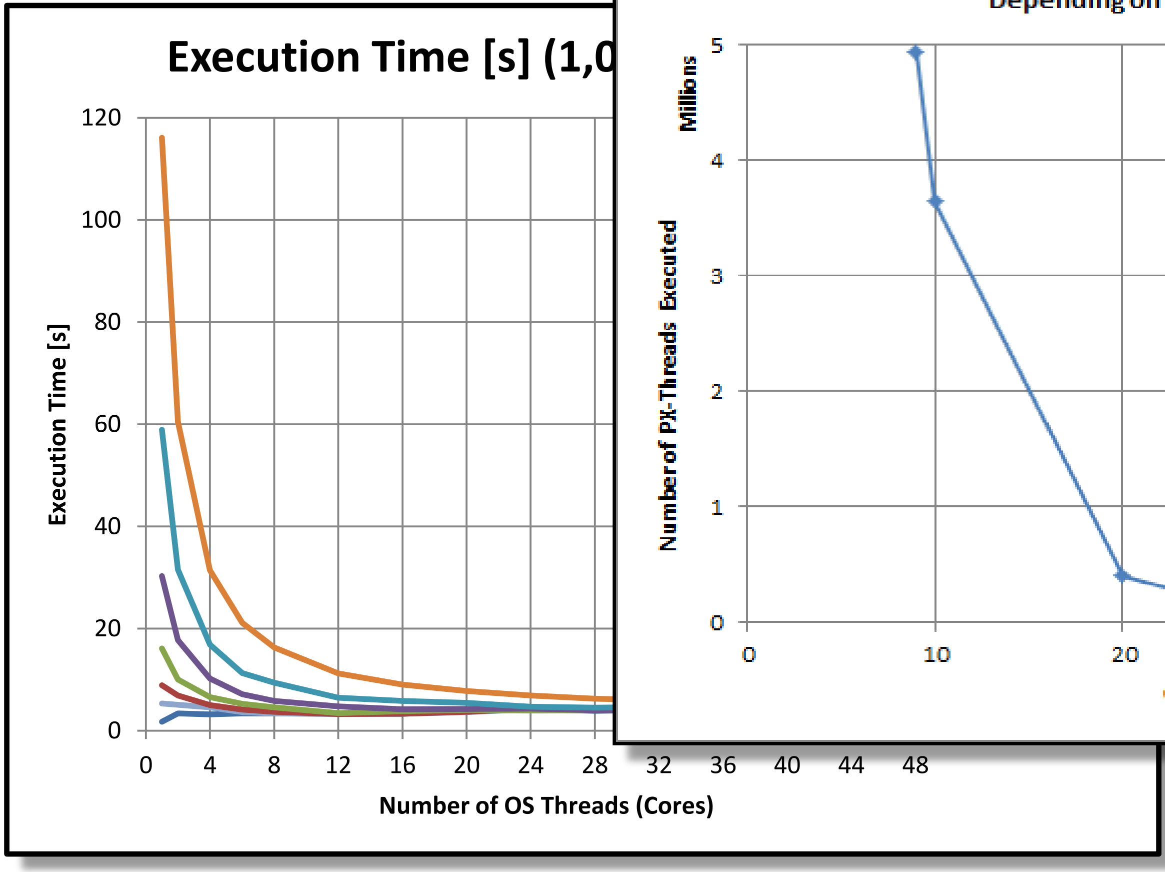


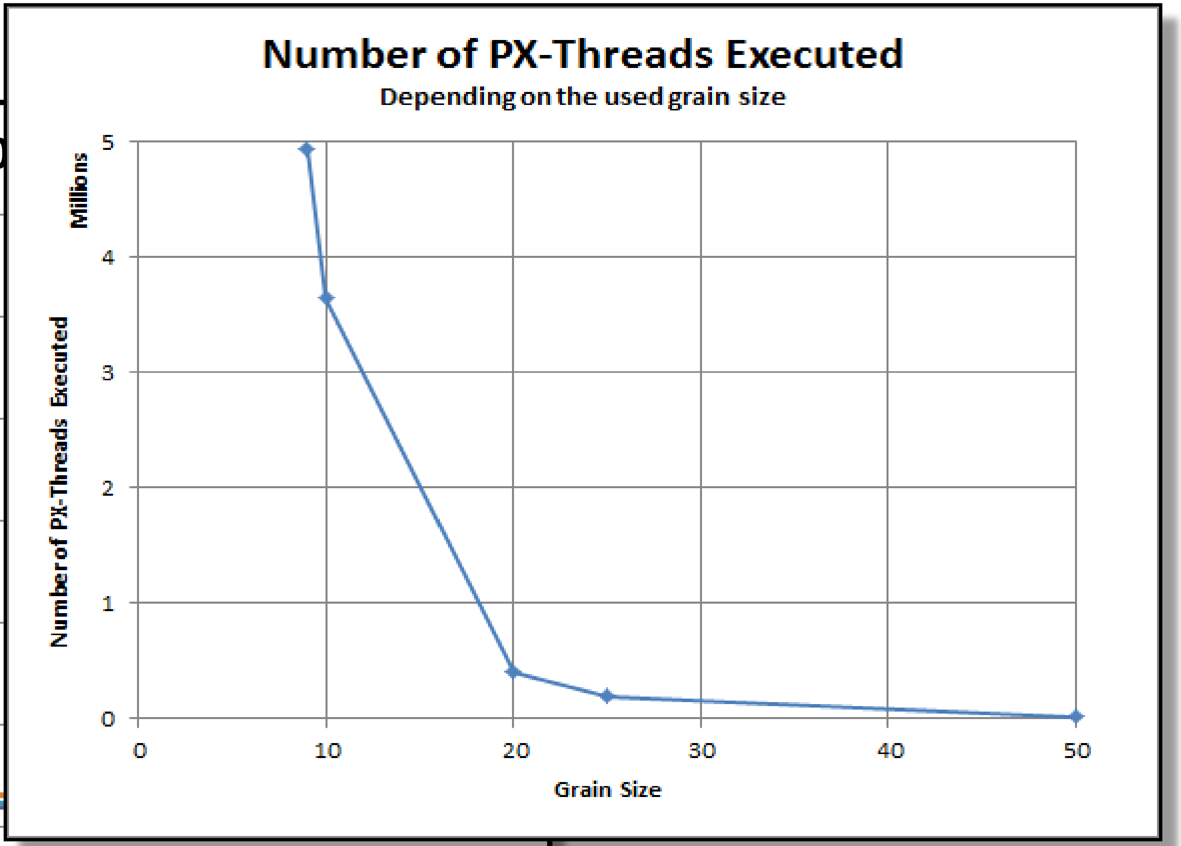
Competing effects for optimal grain size: overheads vs. load balancing (starvation)

Overhead: Threads

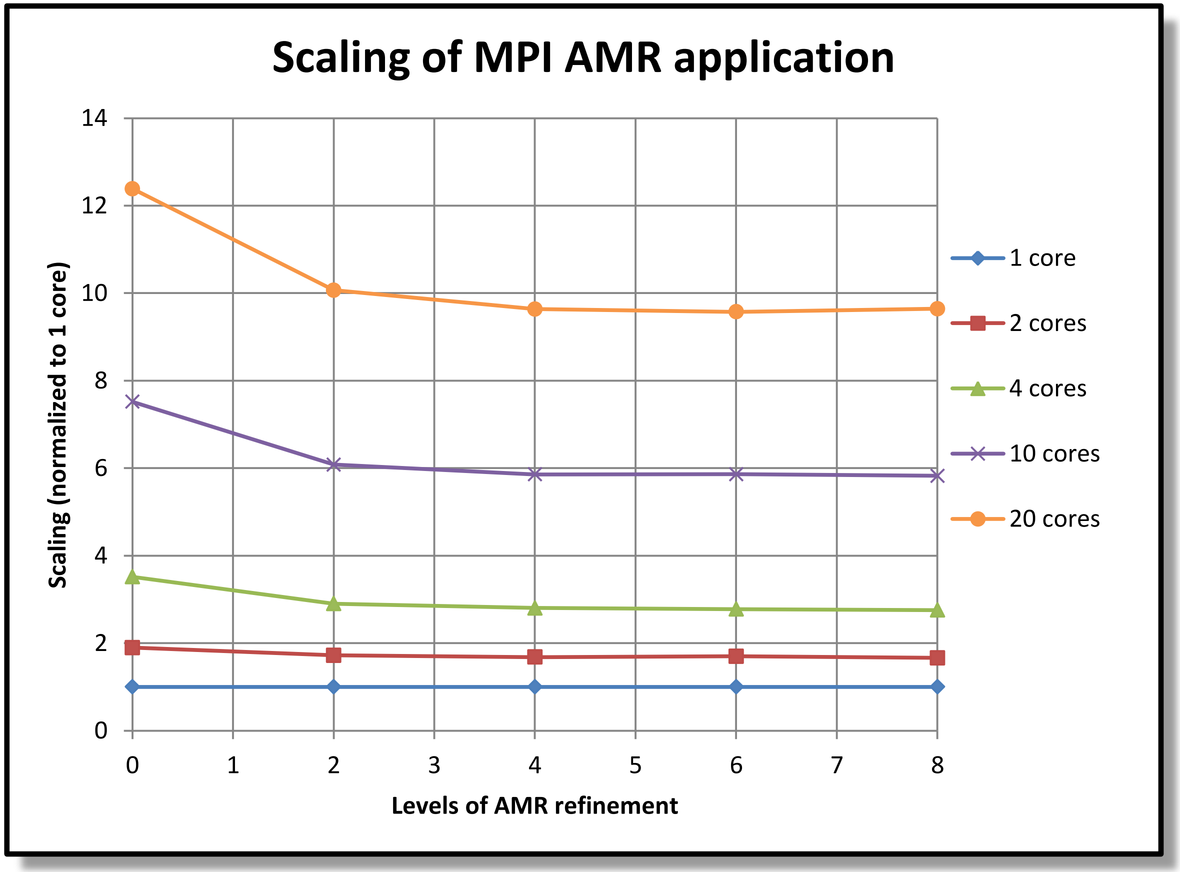


Overhead: Threads

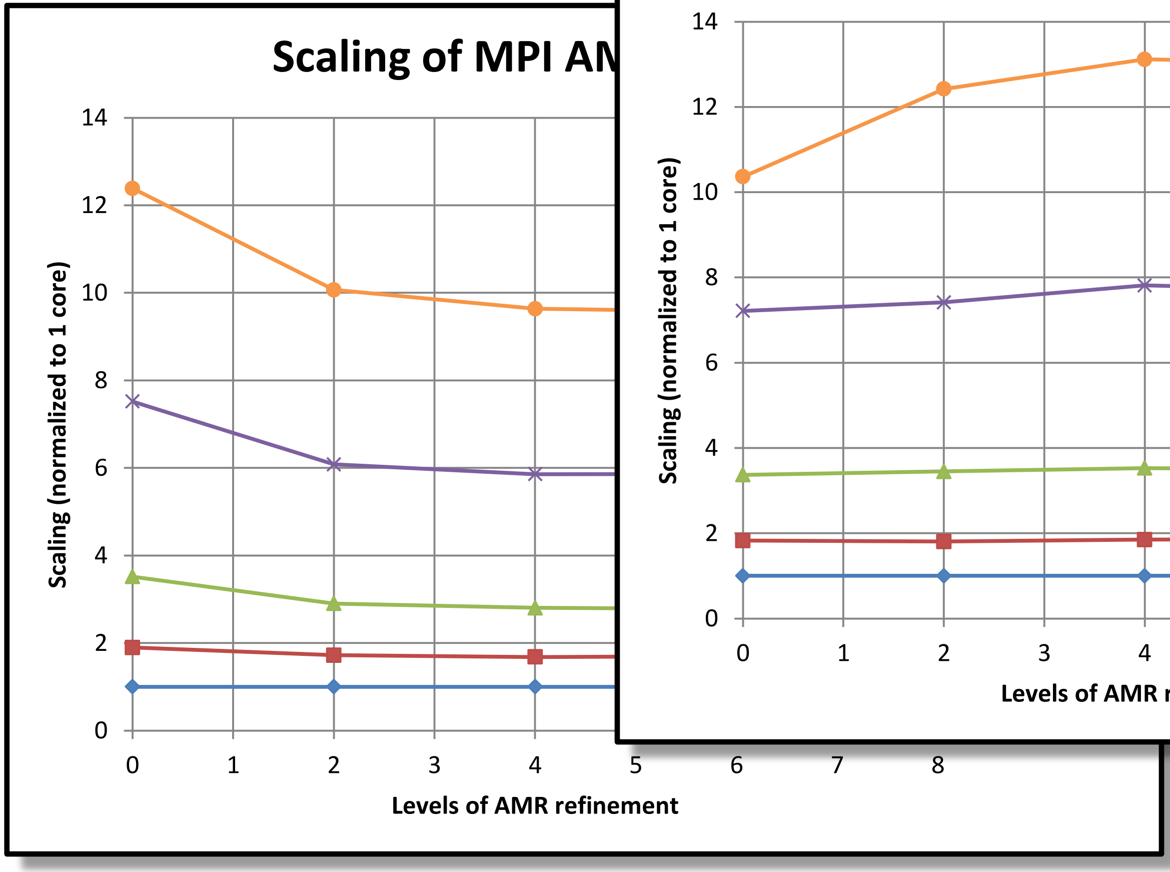


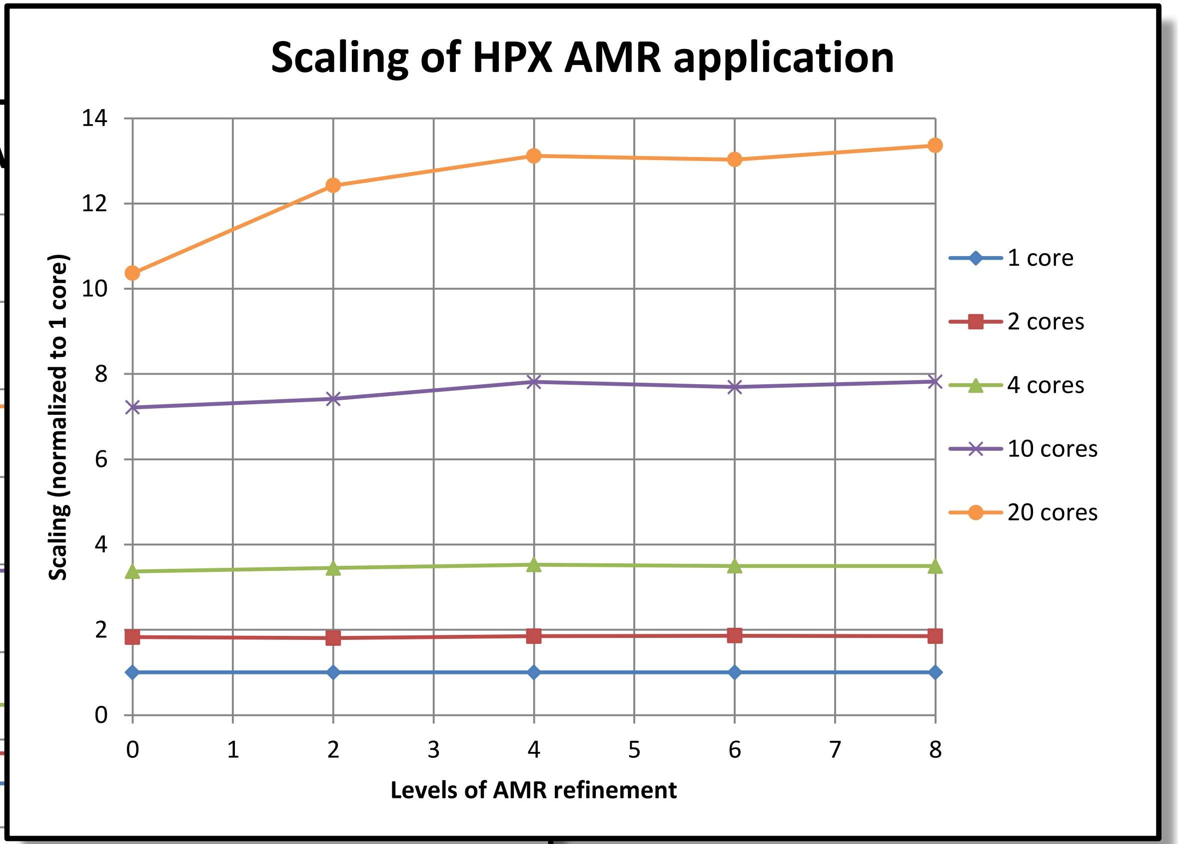


Scaling: AMR using MPI and HPX

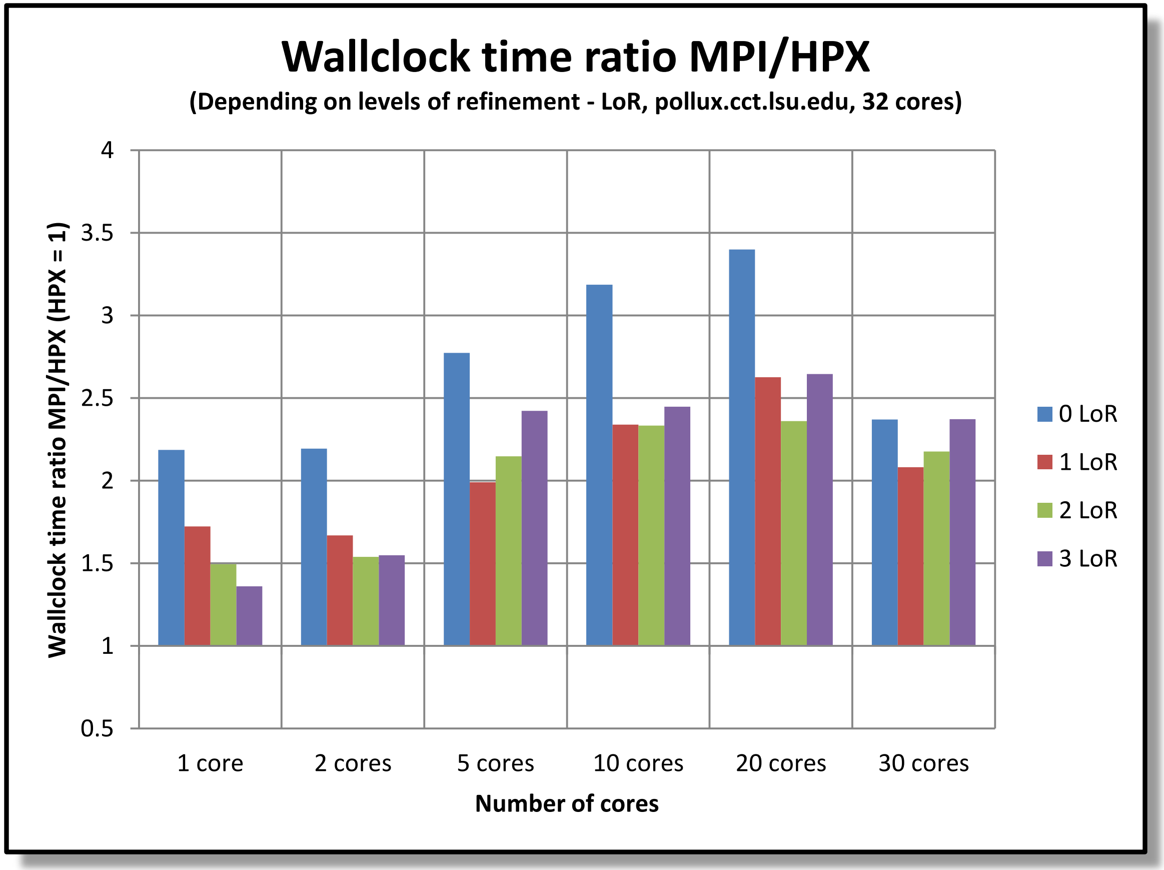


Scaling: AMR using MPI and HPX





Performance: AMR using MPI and HPX



ParalleX

A Cure for Scaling Impaired Parallel Applications ?

ParalleX – Is it a Cure?

- Not completely sure yet
  - Half way through
  - Promising results on SMP systems
  - First (promising) results on distributed Systems
- No code changes required!
- Current projects
  - Custom hardware (FPGAs) accelerating systems functionality
  - Improving performance of AGAS, Parcel transport, …
  - Redefining I/O

ParalleX – Is it a Cure?

- ParalleX execution model can be implemented without adding significantly more overhead than what MPI does
- Implicit load balancing for AMR simulations based on finer grained parallelism highly beneficial
- There are regimes and applications that can benefit from this highly parallel model
- Runtime granularity control is crucial for optimal scaling