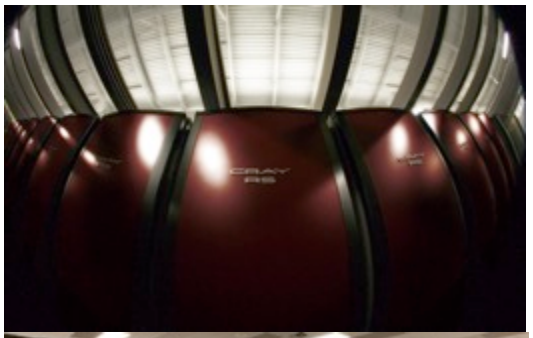


XPRESS: eXascale PRogramming Environment and System Software

Ron Brightwell

Coordinating PI

Sandia National Laboratories

Exceptional service in the national interest

X-Stack PI Meeting

April 6-7, 2016

U.S. DEPARTMENT OF ENERGY

NNSA

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

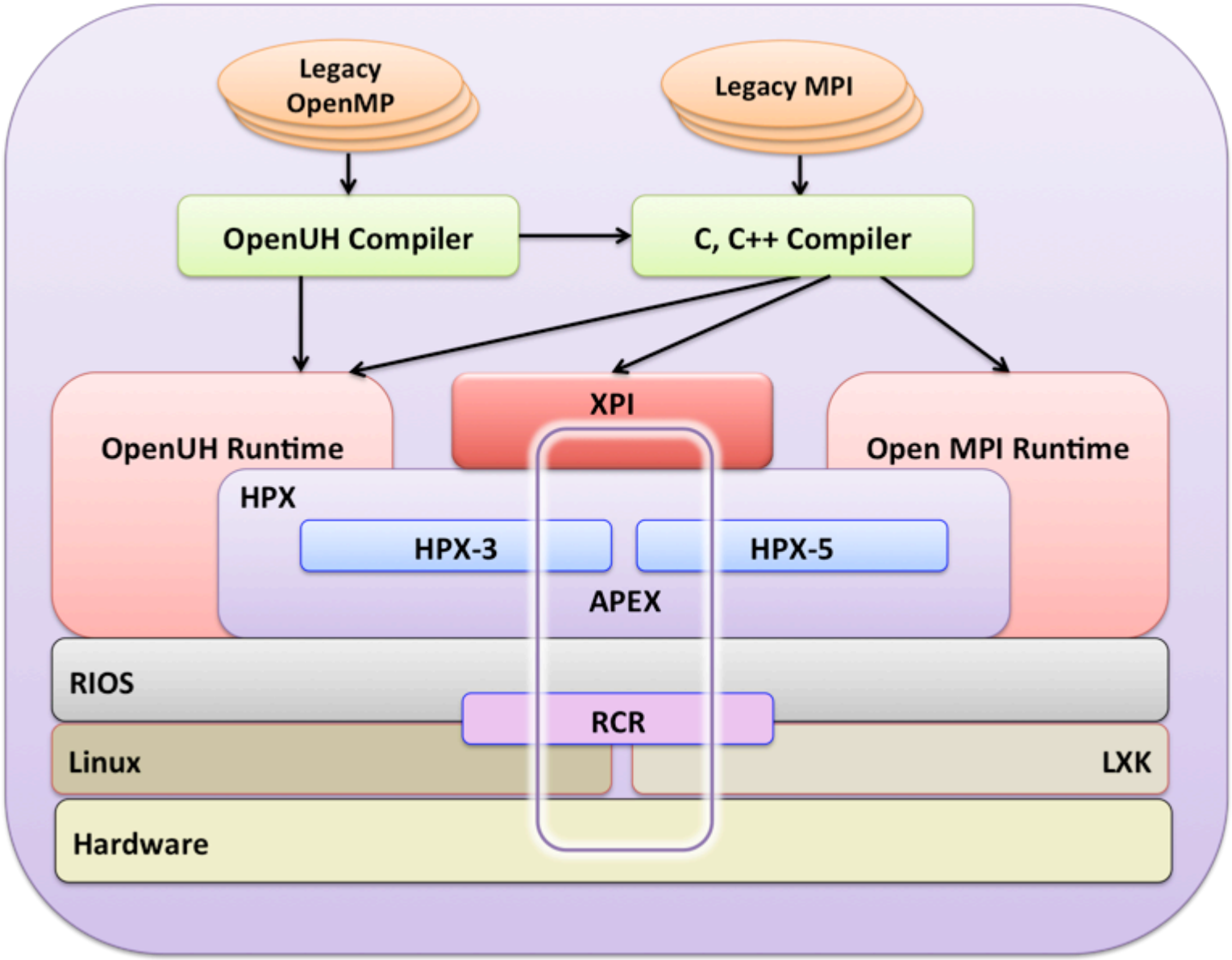
XPRESS Team

Sandia National Laboratories
Indiana University
Lawrence Berkeley National Laboratory
Louisiana State University
Oak Ridge National Laboratory
University of Houston
University of North Carolina at Chapel Hill
University of Oregon

Project Goal

- Prototype implementation of software stack (OpenX) to support the ParalleX Execution Model
  - HPX runtime system based on the ParalleX execution model that supports dynamic resource management and task scheduling
  - LXK lightweight operating system based on the Kitten OS that exposes critical resources to HPX runtime system
  - Runtime Interface to OS (RIOS) definition and description of the interaction between HPX and LXK
  - Support for legacy MPI and OpenMP codes with OpenX

XPRESS Programming Environment Components



XPRESS System Architecture (OpenX)

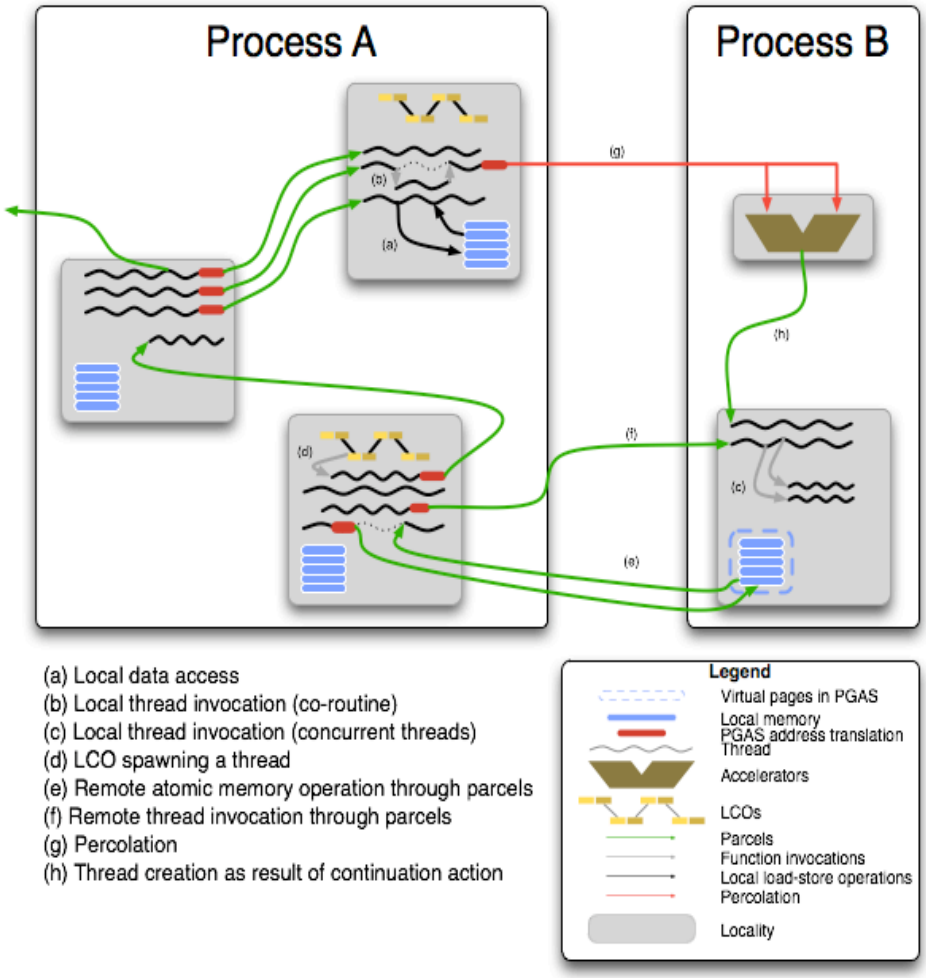
- ParalleX – execution model
  - Indiana University
  - Cross-cutting execution model of system codesign
- LXK – Lightweight eXtreme-scale Kernel (Kitten)
  - Sandia National Laboratories
  - Fourth-generation scalable compute node operating system
- HPX runtime system software
  - Louisiana State University, Indiana University
  - Supports introspection for guided computing through dynamic adaptivity
- APEX, RCR – application introspection
  - University of Oregon, RENCI
  - A derivative of Tau instrumentation and monitoring software system
  - Integration of low-level system data acquisition
- RIOS – Runtime Interface to the OS
  - Interface between the operating system and the runtime system
- Conventional Programming Interfaces for legacy interfaces and applications
  - University of Houston, SUNY Stony Brook
  - MPI, OpenMP

Key Differentiators of Our Approach

- Guided by the ParalleX holistic execution model
  - Provides a revolutionary programing environment for supporting irregular, and time-varying applications
  - Provides a framework for exploring key resource management challenges
    - Dynamic, adaptive runtime system design
    - Integration of performance instrumentation and control
    - Operating system design
- Spans the entire software stack
- Support for legacy programming models and applications

Concept Formulation - ParalleX Execution Model

- Lightweight complexes (threads)
  - Partially ordered operations
  - Guaranteed local register sets
- Message-driven computation
  - Move work to data
  - Keeps work local, stops blocking
- Constraint-based synchronization
  - Declarative criteria for work
  - Event driven
  - Eliminates global barriers
- Data-directed execution
  - Merger of flow control and data structure
- Shared name space
  - Global address space
  - Simplifies random gathers



Technical Strategies

- Execution model for cross-cutting system co-design
  - Unified model of variable parallelism semantics for scalability
- Lightweight kernel for scalable efficient resource management
- Exploitation of runtime information and control
  - Runtime system software
- Introspection
  - Hardware and OS status monitoring
  - Application runtime status and progress towards goal (VMG)
  - Application informed policies
- Dynamic adaptive computation for load balancing
- Active Global Address Space
- Low-level network transport layer for efficient remote task creation
- Interoperability and incremental extensions of common interfaces
- Application properties emphasizing irregular & time-varying

Programmatic Strategy

- Experimental software system
- Full systems stack
- Applications driven studies
- Quantitative evaluation of efficiency and scalability
- Implications for hardware architecture
- Interoperability with existing practices, systems, and libraries
- Innovation where required, incrementalism where sufficient
- Preparation for production deployment
- Leverages prior and ongoing results from other sponsored research providing synthesis and value added
- Engagement of breadth of expertise from labs and academia
  - Avoid myopic paths limited by narrow perspectives
  - Exploit breadth of experiences across sub-disciplines

PARALLEX EXECUTION MODEL

Formal Semantics for ParalleX

- Problem
  - Exascale execution models are large specifications defining the delicate interrelations of their component layers and governing their interoperability. Defining execution models is error prone, may leave undetected inconsistencies and may be incomplete. Furthermore, execution models are hard to communicate effectively to programmers, and researchers.
  - How do I know whether my execution model is well designed?
- Solution

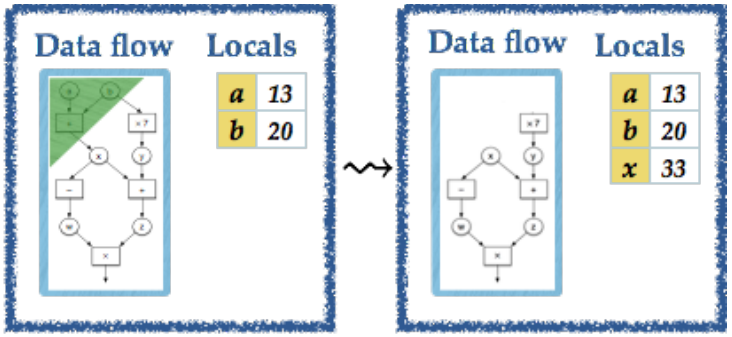
  - Formal semantics allows for:
    - A precise description of the execution model
    - Detecting design mistakes early and ensuring the completeness of the specified behavior
    - Communicating the model effectively
    - Opening the possibility of proving programs correct

Formal Semantics for ParalleX

- Recent Results
  - Operational semantics for a core fragment of ParalleX
  - Proposal type system for ruling out data-races
  - Executable prototype implementation of formal semantics

Excerpt from the operational semantics

$$
\begin{array}{ll}
 & M = S(\text{memory}),\ M(a) = c \\
\textbf{if} & z \leftarrow x + y \in c(\text{control}) \\
 & c(\text{locals})(x) = n_x \\
 & c(\text{locals})(y) = n_y \\
\textbf{then} & S \longmapsto S' \\
\textbf{where} & \\
 & S' = S\{\text{memory} \mapsto M\{a \mapsto c'\}\} \\
 & c' = \{\text{control} \mapsto c(\text{control}) - \{z \leftarrow x + y\}, \text{locals} \mapsto L'\} \\
 & L' = c(\text{locals})\{z \mapsto n_x + n_y\}
\end{array}
$$

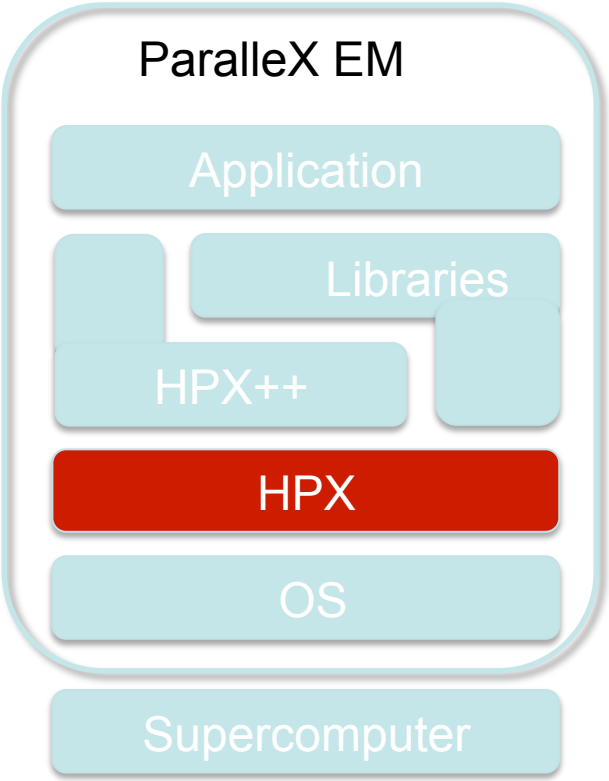


- Impact

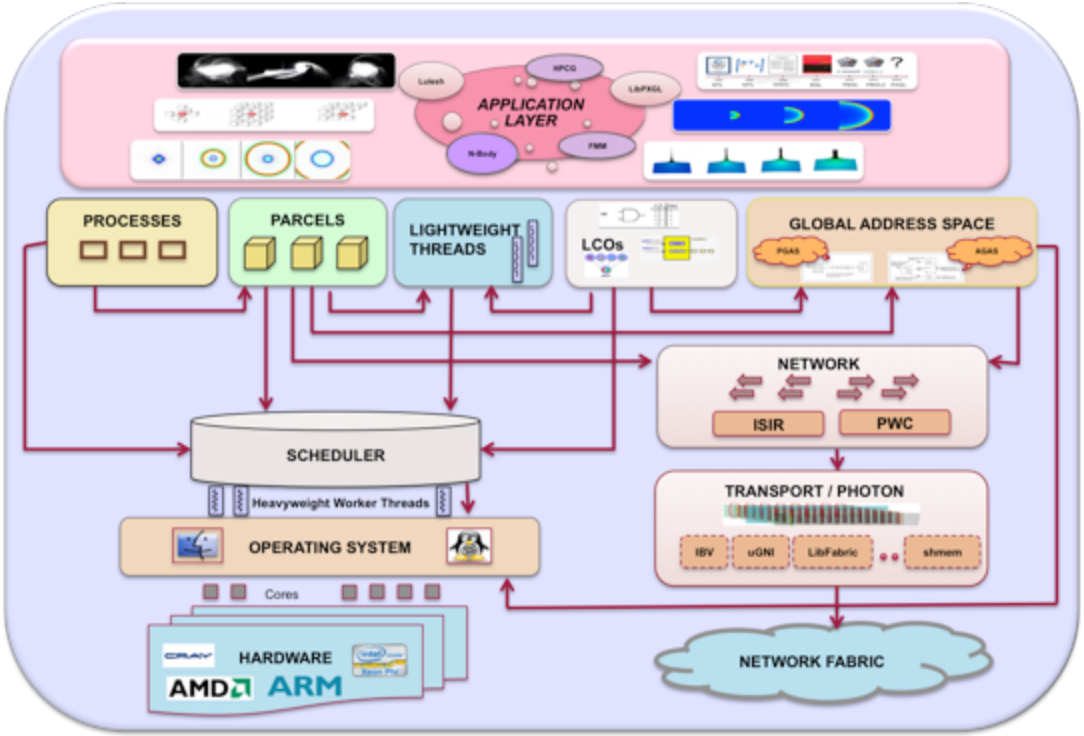
The shift to eXascale computing promises a high impact on science and industry. Formal Semantics will place eXascale computing on a firm foundation, enabling a more rapid and confident shift to eXascale built upon a reliable and robust infrastructure.

HPX-5

Development and Extension - HPX-5

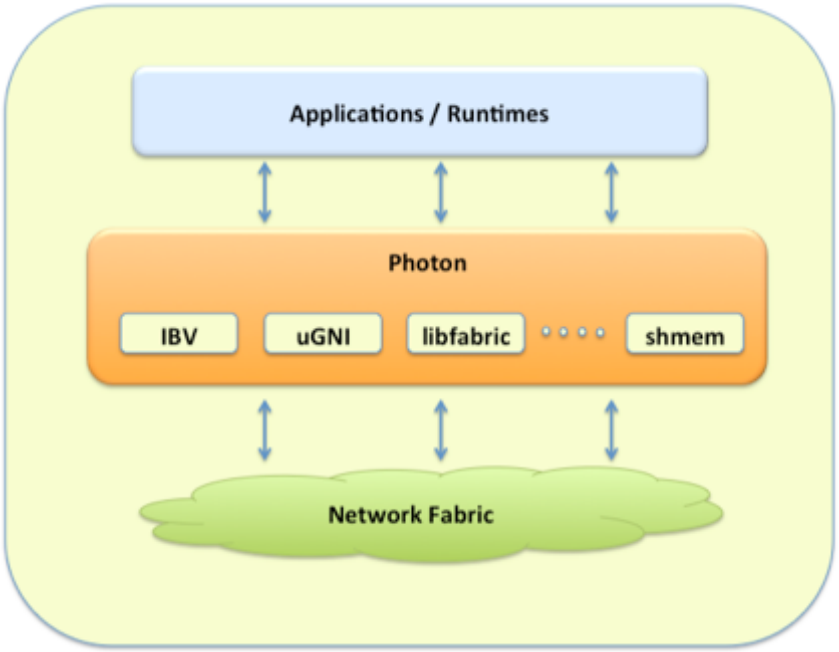


HPX-5 is an approximation of the ParalleX execution model

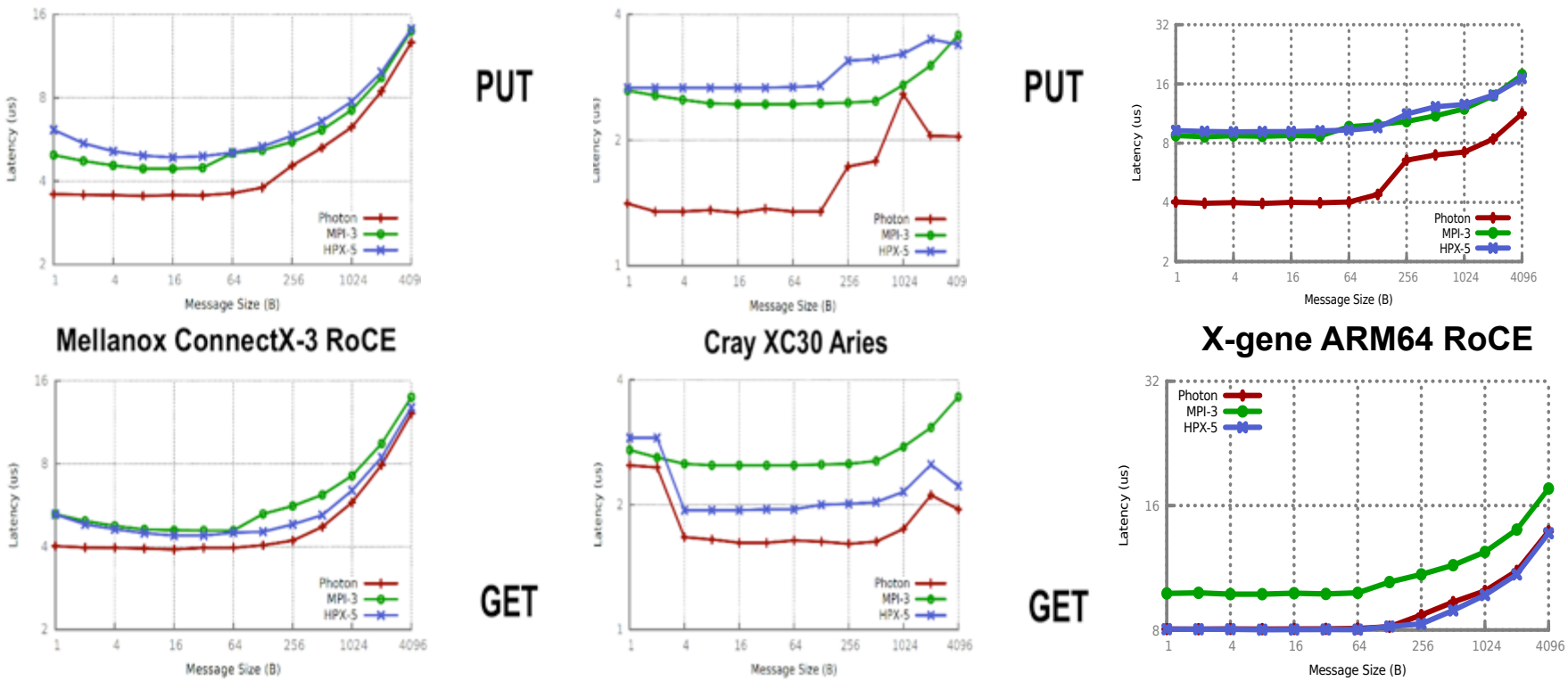


Integrated Communication Library

- Problem
  - Support a tight coupling of the runtime system with the underlying network fabric that scales and remains performant in eXascale environments
- Solution
  - Photon abstracts RDMA libraries across systems and integrates with HPX-5 to support 1-sided asynchronous networking with a put-with-completion (PWC) model
  - Network completion events drive interrupt-style actions within HPX-5 to reduce overhead and latency
- Impact
  - Photon in HPX-5 demonstrates improved performance for application with a modular design to support future generation networks



Integrated Communication Library



HPCG using HPX-5 – Various Approaches and Plots

Parcels approach

```
rank = malloc Comm Size
init(rank)
for i from 0 to Iterations
  for phase in Phases
    for neighbor in Neighbors(rank, phase)
      recvs += IRECV(rank, phase, neighbor)
    for neighbor in Neighbors(rank, phase)
      Pack(rank, phase, neighbor)
      ISEND(rank, phase, neighbor)
    WAIT_ALL(recvs)
    unpack(rank, phase, neighbor)
    parallel_for(computation(rank, phase))
```

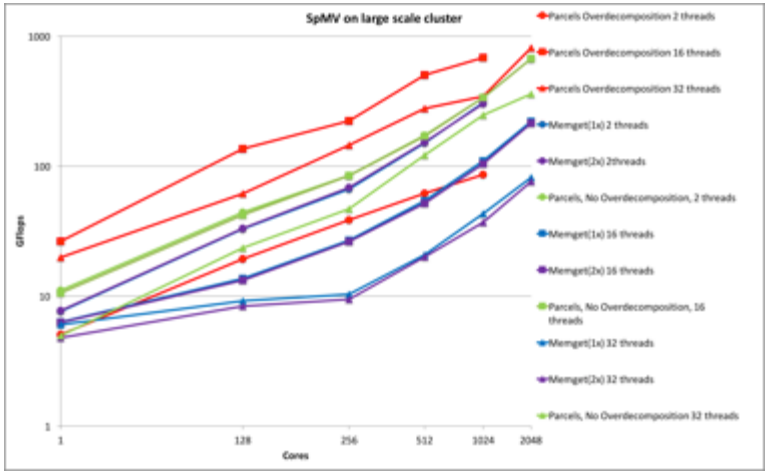
RDMA memget approach

```
rank = malloc Comm Size
init(rank)
for i from 0 to Iterations
  for phase in Phases
    set_lco(rank)
    parallel_for xRow in NeighborXRows(rank, phase)
      memget(xRow,lco[xRow])
    parallel_for(computation(RowsOfA[rank], phase))
    wait(xRowsLCO)
where
    on access to a remote rows of x while working on a row
    1) spawn a thread to finish computation
    2) on access to to any remote xRow: wait(lco[xRow])
    3) after computation is finished: set_lco(xRowsLCO)
```

RDMA memput approach

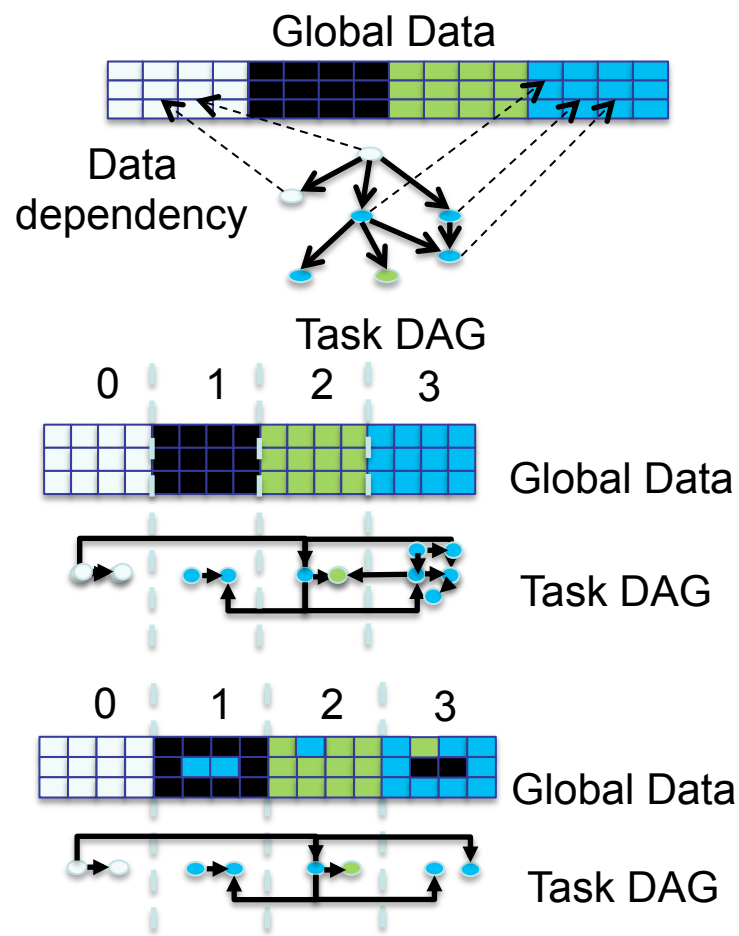
```
rank = malloc Comm Size
init(rank)
for i from 0 to Iterations
  for phase in Phases
    for neighbor in Neighbors(rank, phase)
      memput(neighbor,lco[neighbor])

    wait(lco[rank])
    parallel_for(computation(rank, phase))
```



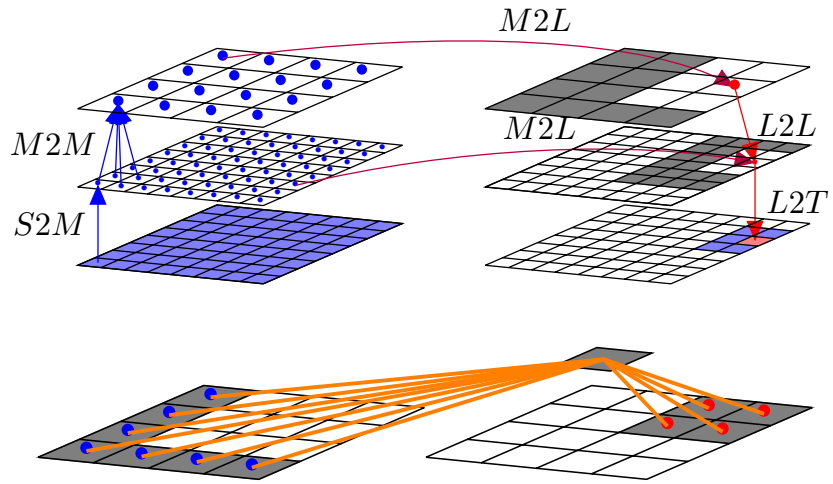
Active Global Address Space (AGAS)

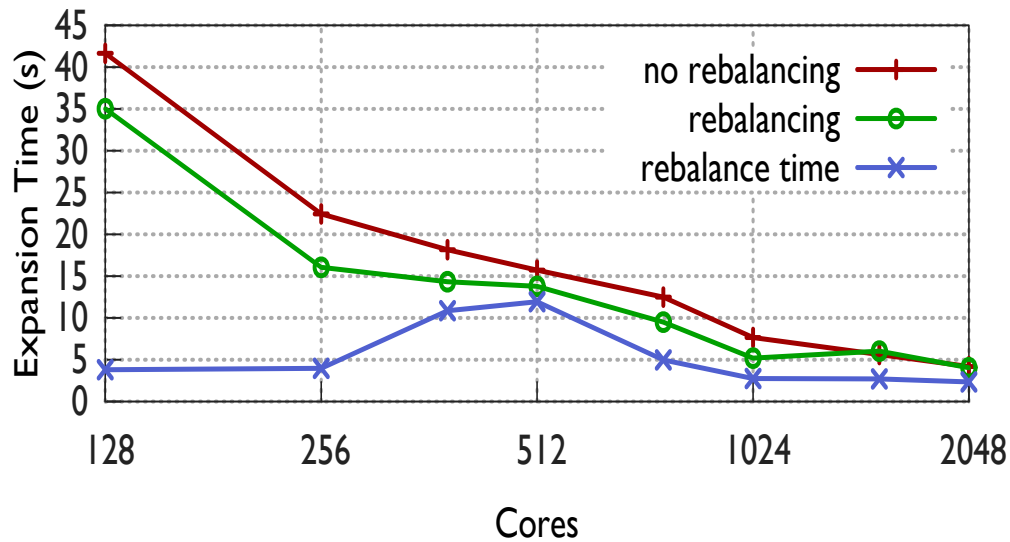
- Problem
  - Dynamically varying load in the execution of adaptive applications leads to uneven system utilization and consequently limits scaling of such applications
  - Active migration of data at runtime in the global address space is challenging
- Solution
  - The active global address space (AGAS) allows on-the-fly relocation of global data between different physical localities
  - Inflight parcels targeted to moving blocks are forwarded appropriately
  - Optimal global data redistribution is determined through communication graph partitioning



Active Global Address Space (AGAS)

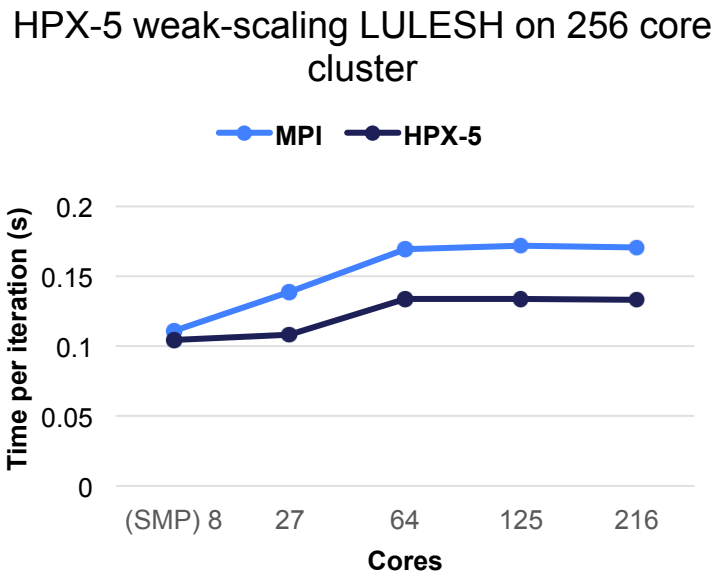
- Problem
  - N-body like problems appear in many scientific applications. The naïve solution has $O(N^2)$ complexity, whereas the Fast Multipole Method(FMM) reduces this to O(N) complexity up to any prescribed accuracy requirement
- Impact
  - AGAS in HPX-5 enables asynchronous load balancing in FMM through active migration of nodes in the global spatial decomposition tree

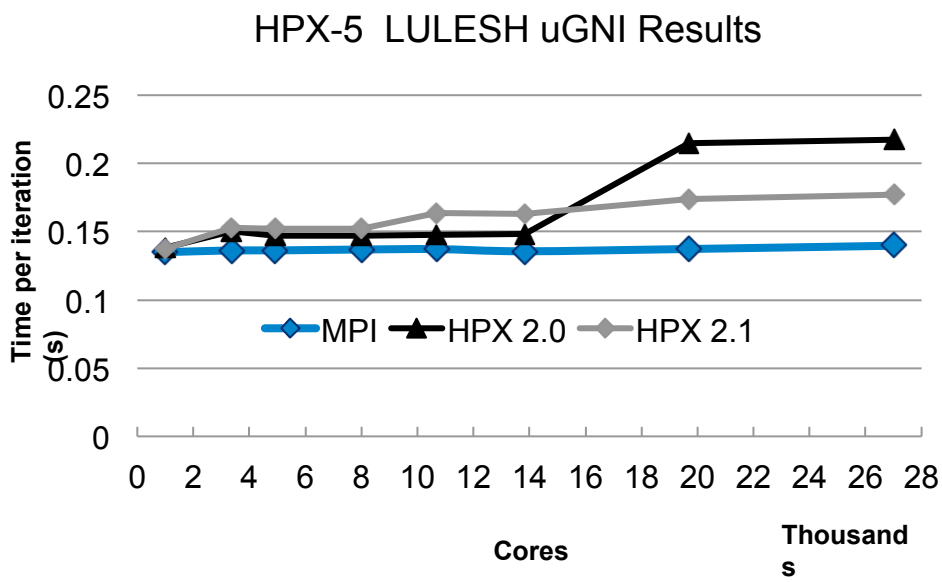




Demonstration of Scalability and Performance

- Problem
  - Demonstration of exascale performance is not possible on current hardware
- Solution
  - Demonstrate Scalability on NERSC Edison (concurrency), while showing highly-threaded improvements on testbed systems such as NERSC Babbage and local clusters
- Impact
  - Testing shows HPX will provide DOE codes with a path forward

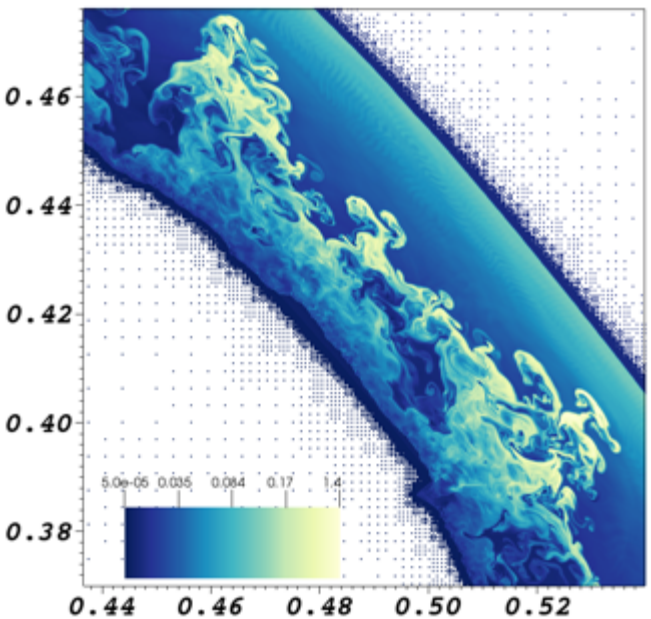


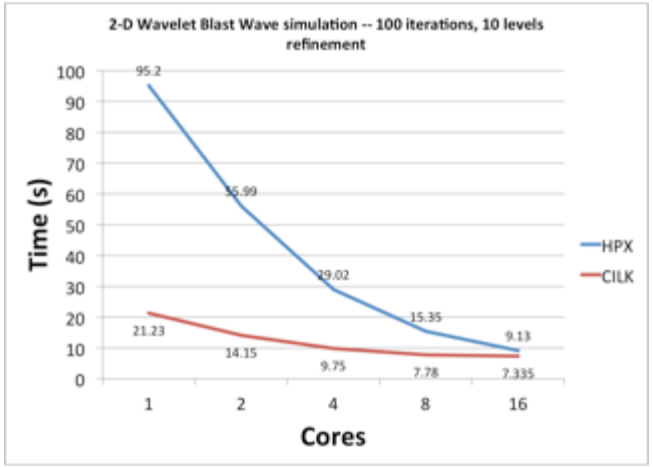


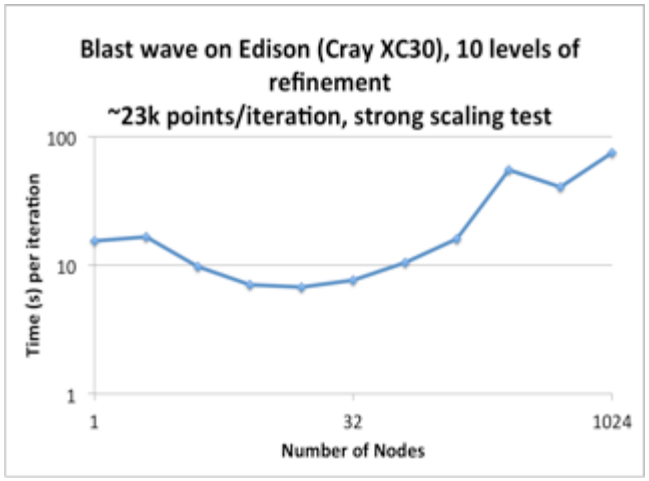
Demonstrations of Scalability and Performance: Wavelets

- Problem
  - Dynamic, irregular, non-uniform applications are scaling constrained using conventional parallelization techniques
  - Applied to hydrodynamics:

    arXiv:1512.00386
- Solution
  - Asynchronous Multi-Tasking (AMT)

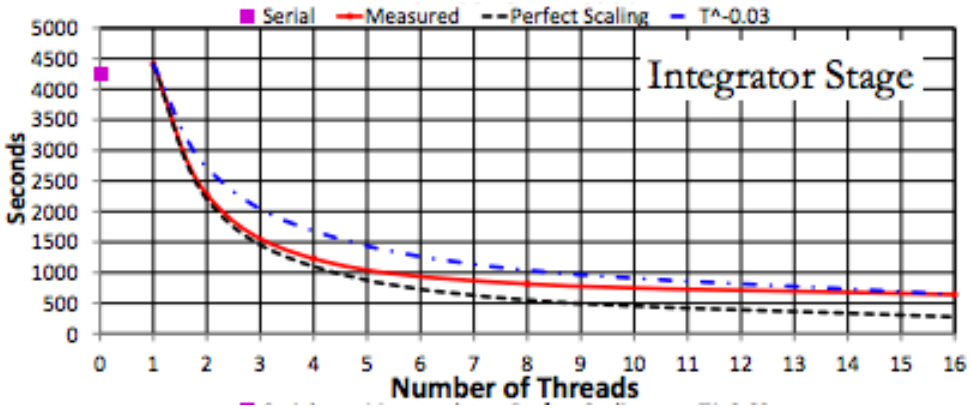




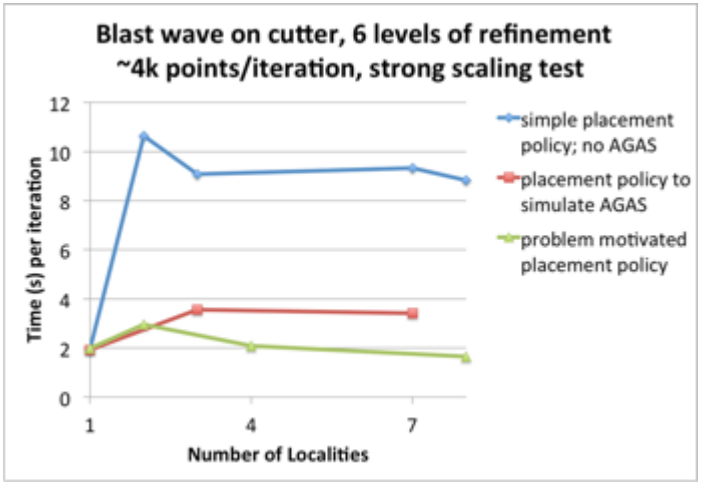


Demonstrations of Scalability and Performance: Wavelets

- Asynchronous Multi-Tasking (AMT)

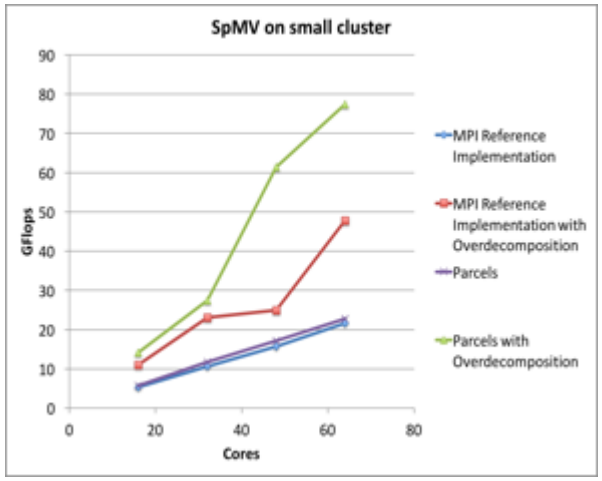


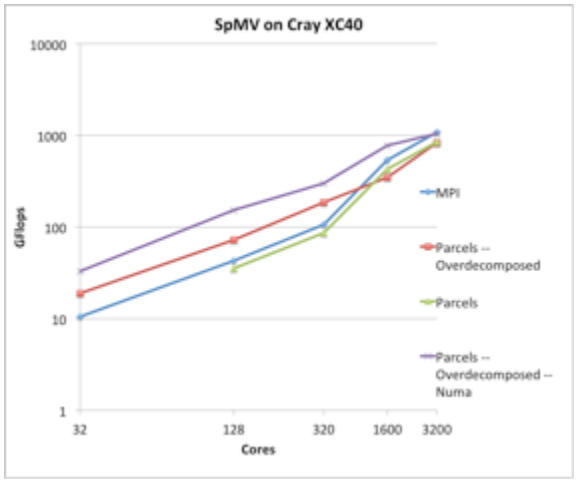
- Exploring AGAS

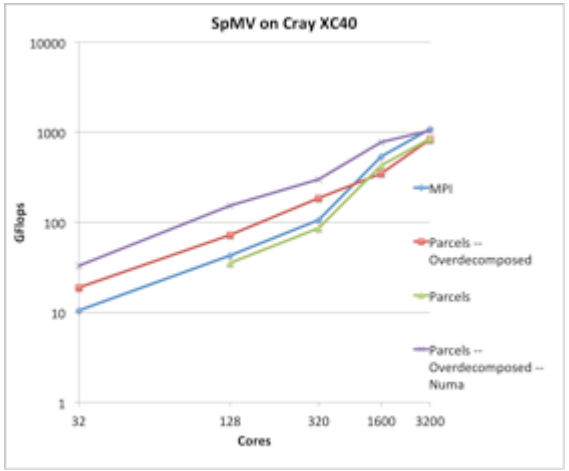


HPCG using HPX-5

- Problem:
  - High Performance Linpack performance doesn't represent most applications
  - HPCG (High Performance Conjugate Gradient) benchmark only achieves a small fraction Linpack performance
- Solution:
  - Asynchronous multi-tasking runtime systems easily enable one-sided linear algebra operations
- Recent results:
  - The below plot shows performance of SpMV (Sparse Matrix Vector multiplication).



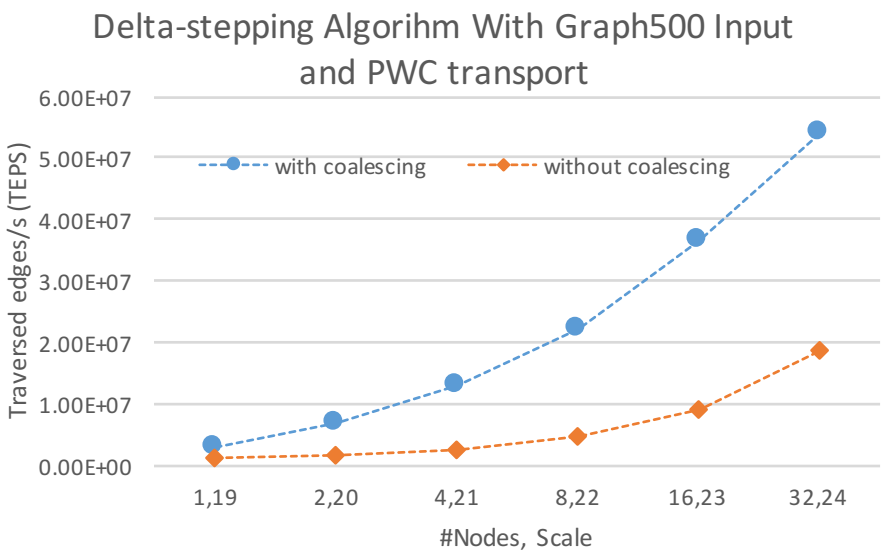




Graph Analytics in HPX-5

- Problem
  - Dynamic, irregular, data-dependent graph analytics applications generate large numbers of small (in order of bytes) messages and extremely (vertex-level) fine-grained parallelism

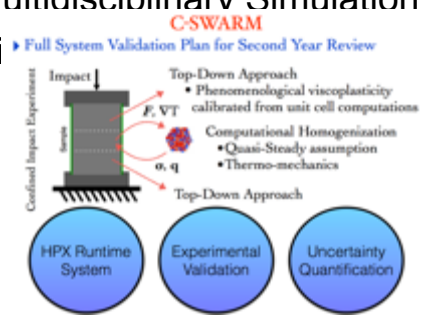
- Solution
  - HPX-5 consistently improves (between versions) the performance of fine-grained parallelism encountered in graph applications
- Recent results
  - Investigation of application driven scheduling to tailor the runtime to application needs and of coalescing of parcels to decrease communication overheads
- Impact
  - Many emerging applications rely on fine-grained data-driven parallelism

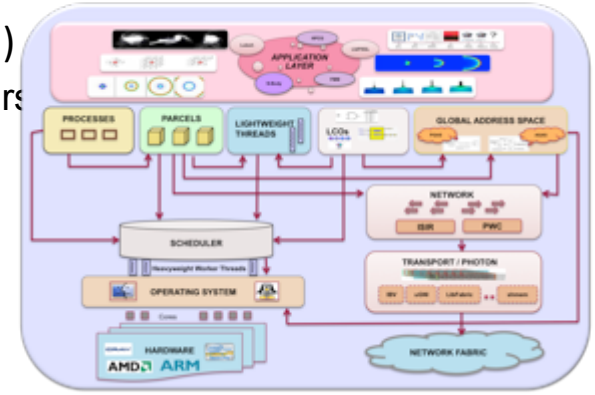


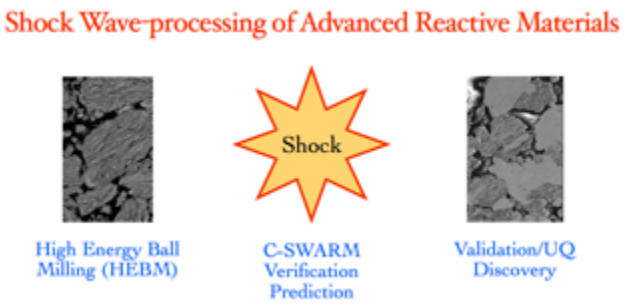
Performance of Delta-Stepping algorithm for Single-Source Shortest Paths (SSSP) executed on Graph500 benchmark inputs with and without coalescing

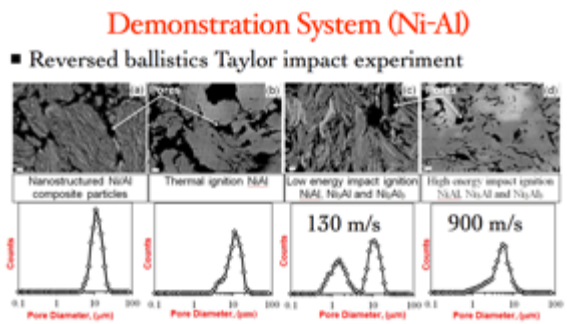
Enhancement and Exploration (external) - Reactive Material Simulations

- Problem
  - Multidisciplinary predictive science
- Solution
  - Predictive ce Academic Alliance Program (PSAAP II)
    - » 3 Multidisciplinary Simulation Centers
    - » 3 Si



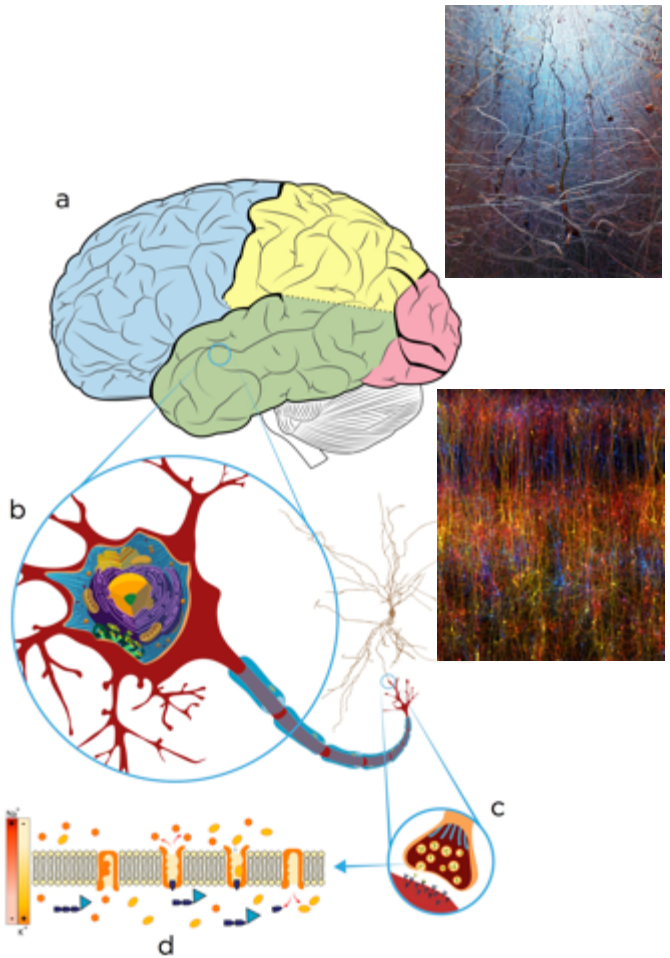






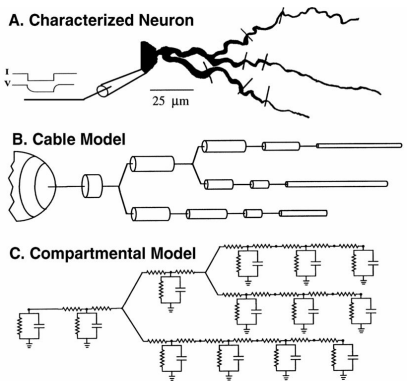
Human Brain Simulation (BlueBrain Project @ EPFL)

- Dynamically adaptive software to allow simulation at different scales:
  - Point neuron level simulation (thousands/millions of neurons per node)
  - Compartmental level simulation (few neurons per node)
  - Biomolecular level simulation (one neuron across several nodes)
- Multirate and variable time-step solvers (based on each different mechanism) reflect better the neuronal networks behavior, contrarily to fixed time-step solvers
  - This requires a totally asynchronous programming paradigm as provided by HPX
- Hide communication and threading complexity
  - Developer only focus on writing the logic; HPX handles parallelization
- Transparent load balancing
  - Task stealing queue allows balancing of work across threads
  - Global Address Space allows memory to move to different localities to balance work across nodes
- Removal of collective communication and computation calls:
  - Simulation should be a *free system* where computation of objects is independent
  - Suitable for simulation of objects with unpredictable execution times



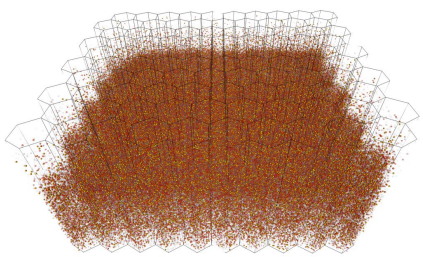
HPX-5 and Brain Simulation

From characterized neuron to compartmental model



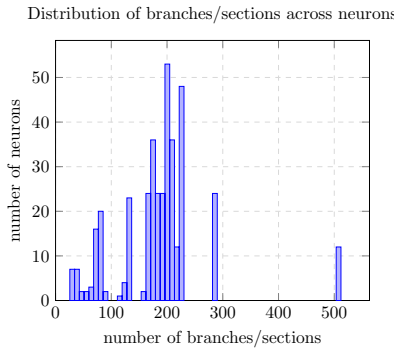
(source: Christof Koch and Idan Segev, Methods in Neuronal Modeling: From Ions to Networks)

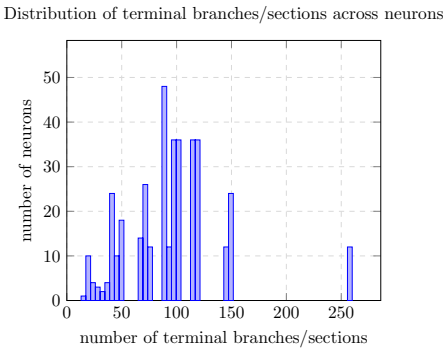
From compartmental model to neural circuits

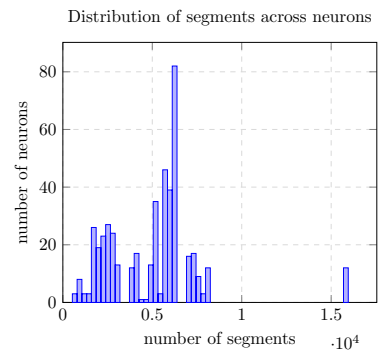


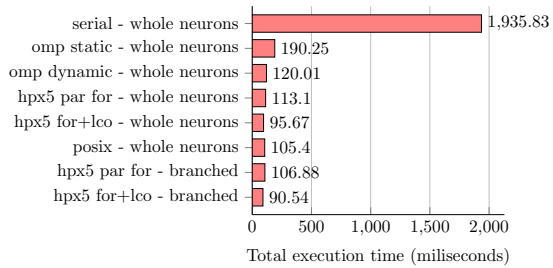
A Hodgkin-Huxley simulation of 3.1M neurons.
(represented as points for simplicity)

mouse brain: 80M neurons; human brain: 100B neurons









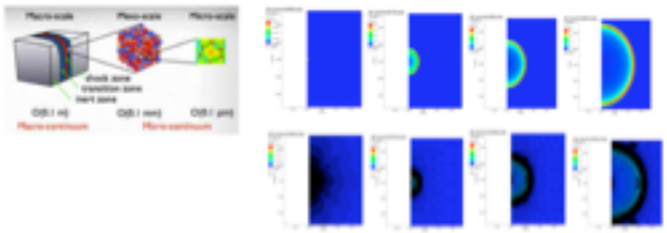
	total time	per neuron time	speed up
serial - whole neurons	1935.83 ms	3.6049 ms	1.0x
omp static - whole neurons	190.25 ms	0.3543 ms	10.2x
omp dynamic - whole neurons	120.01 ms	0.2235 ms	16.1x
hpx5 par for - whole neurons	113.10 ms	0.2106 ms	17.1x
hpx5 for+lco - whole neurons	95.67 ms	0.1782 ms	20.2x
posix - whole neurons	105.40 ms	0.1963 ms	18.4x
hpx5 par for - branched	106.88 ms	0.1990 ms	18.1x
hpx5 for+lco - branched	90.54 ms	0.1686 ms	21.4x

Hardware specs: IBM machine with 40 nodes; Intel(R) Xeon(R) CPU E5-2670 0 @ 2.60GHz; 1 thread per core, 8 cores per socket, 2 sockets, 2 NUMA nodes; L2 Cache 20480 KB; 128 GB RAM; **Launch command:** `mpirun -np 1 --mca btl ^openib --map-by node ./a.out $input-data --hpx-dbg-mprotectstacks --hpx-stacksize=100000`

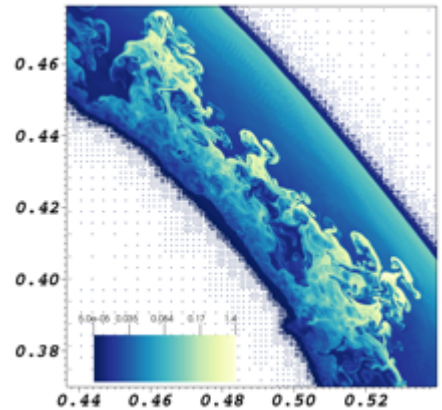
HPX-5 enables unique branched modality for plasticity that is competitive for use in the core neuron benchmark.

Applications using or Trying HPX-5

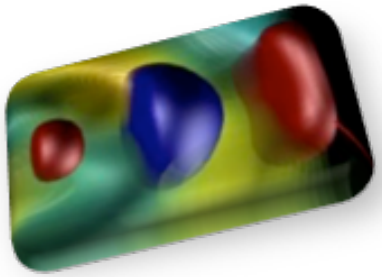
Shock wave processing of advanced materials



DOE NNSA DE-NA0002377 (PSAAP2)



Wavelet methods for fluid research in conjunction with Daniel Livescu (LANL) See also arXiv: 1512.00386



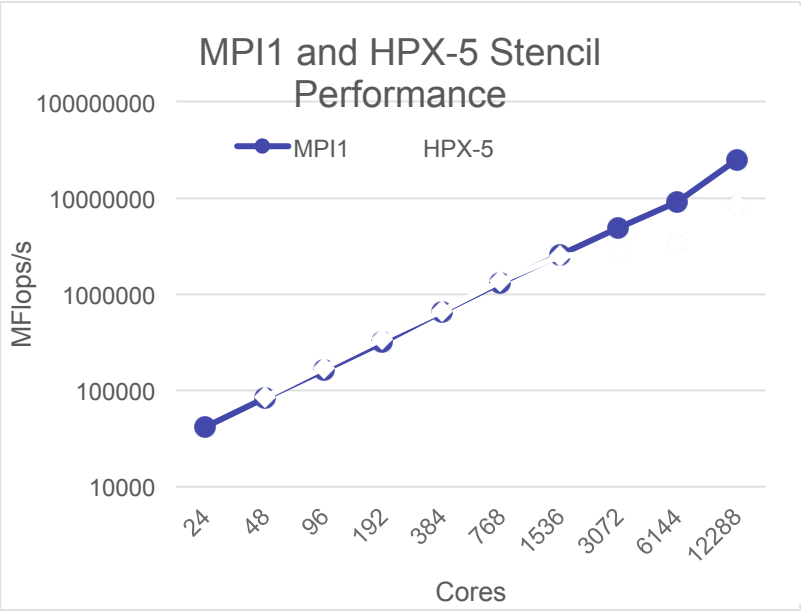
PICSAR: Laser Driven High-Energy Density Plasma and Accelerator Technology

DSL for linear algebra through
DOE NNSA DE-NA0002377 (PSAAP2)

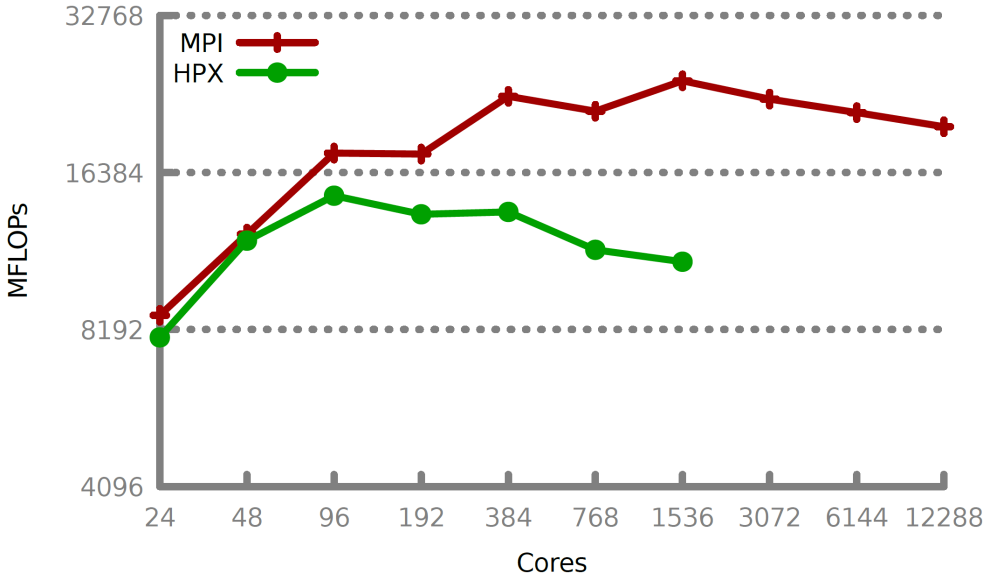
HPX support for LULESH through
DOE DE-SC0008809

Intel Parallel Research Kernels

- Performance plots of Stencil, Synch_p2p – which underlie a wide range of computational science applications

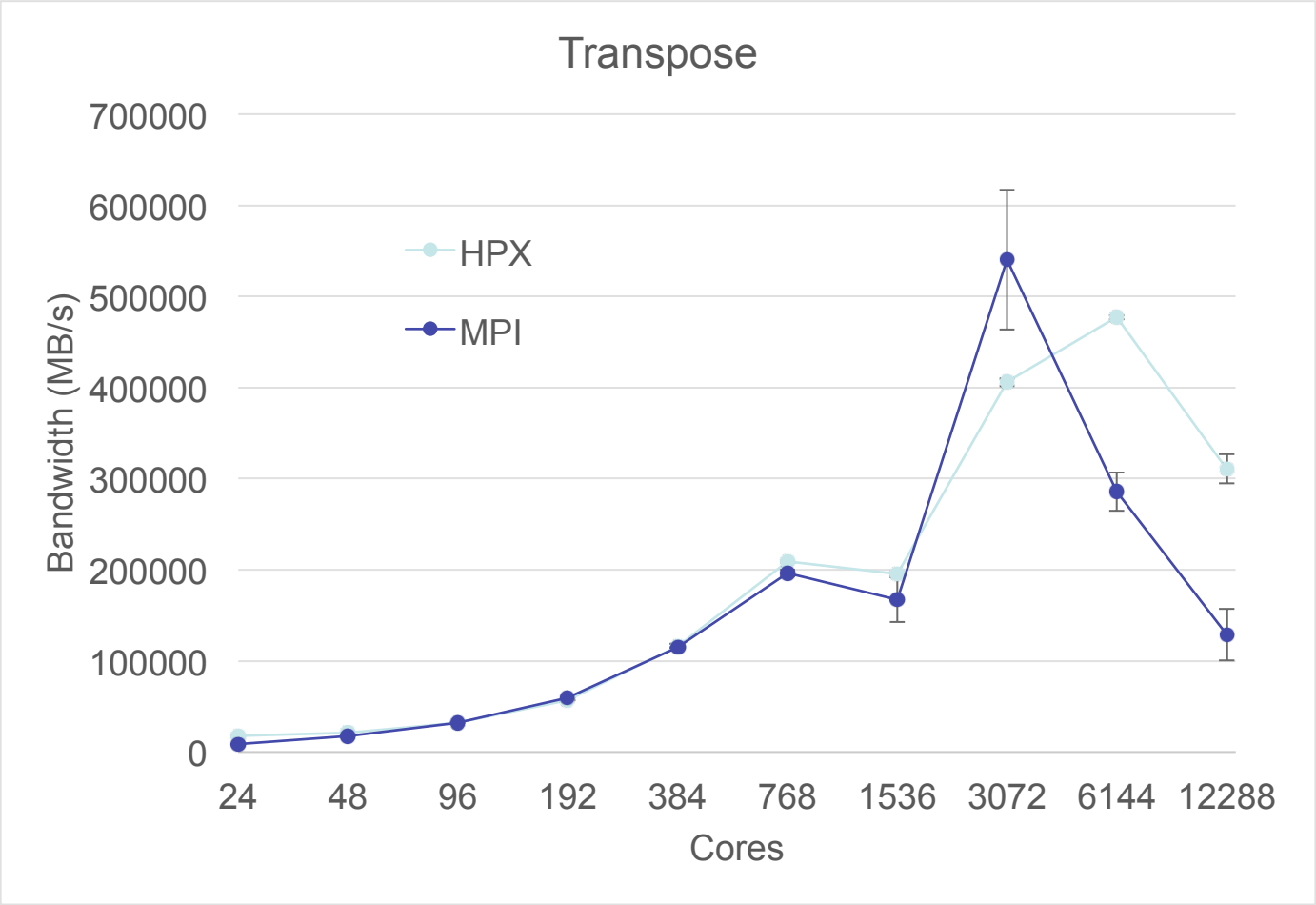


Stencil kernel



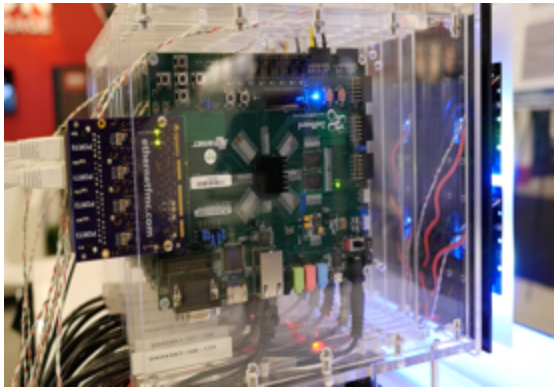
Sync_P2P kernel

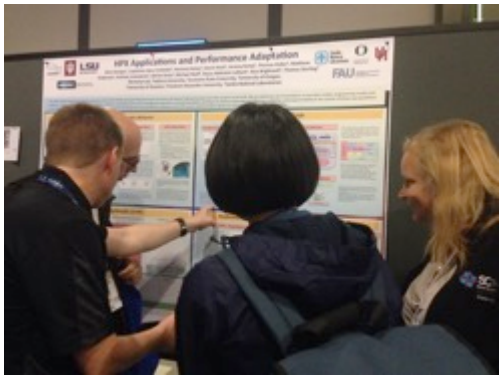
Intel Parallel Research Kernels



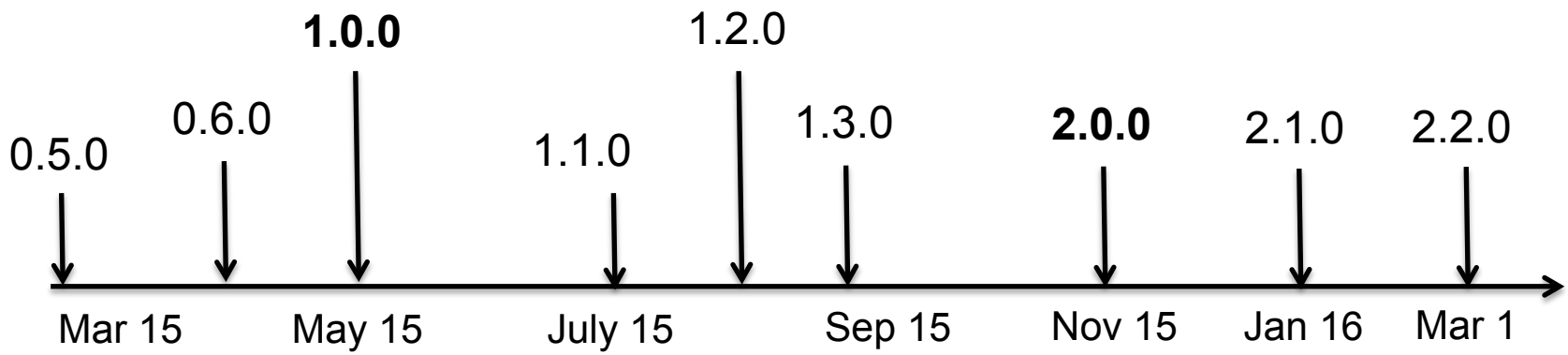
Performance plot of transpose Intel kernel

Early Popularization – Technology Demonstrations





Incremental Delivery



- Active Development and Regular Release Cycles
- Open-source Agile development
  - Available at: http://gitlab.crest.iu.edu
  - Nightly regressions and performance testing on 6+ supercomputers
- Seeking community engagement
  - Runtime development, tools support, more applications

Evaluation Plans for Software Prototypes

- ParalleX formal specification completed and analyzed for correctness and completeness; available for software compliance
- Completion of ParalleX feature set in HPX-5 implementation
- Release of HPX-5 runtime and Photon transport layer on OpenHPC consortium
- Deploy HPX-5 on diversity of platform types and scales to establish robustness of deployability, robustness, and scalability
- Derive measurements of runtime mechanism overheads by means of synthetic micro-benchmarks
- Select and port representative applications for agency mission-critical and end-science applications to measure and compare with baseline control cases; analyze sensitivities for projected exascale operational behavior
- Validate that known and adopted programming interfaces can be fully supported in principle by HPX-5 or determine imposed inadequacies
- Work with Rice University's Habanero-C and SNL DARMA to expand utility of test cases.

Technology Transition Opportunities

- Share with industry partners
  - Current: Cray, Intel, Micron
  - Possible: AMD, ARM, HPE
- Deploy and maintain HPX-5/Photon on OpenHPC.
  - IU is founding member and member of Linux Foundation
- Update introspective policies interface for users and compiler
- Begin tutorials starting in 2017
  - Tutorial documentation currently available
  - Online on-demand video MOOC to be developed and released 2016
- Publish monograph with MIT Press
  - Invited to submit proposal; discussed with publisher editors
- Extend to commercial software

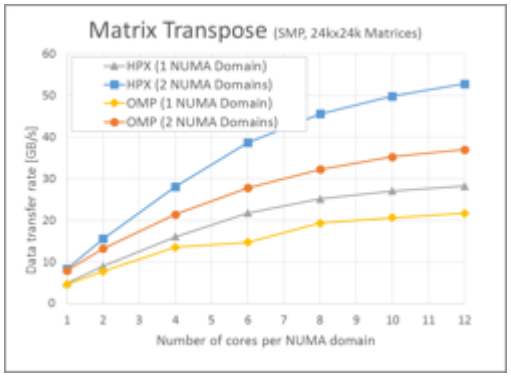
Lessons Learned

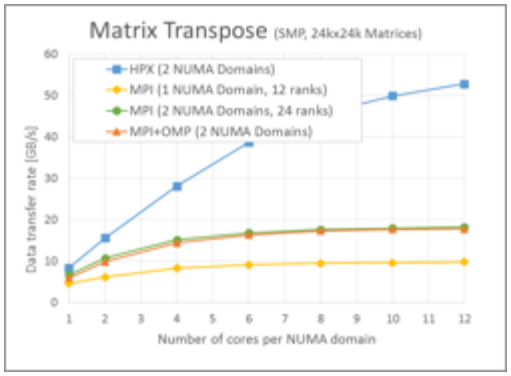
- Dynamic adaptive computing methods exploiting runtime application and system state information offers opportunity for potentially significant improvements in efficiency and scalability while improving user productivity and performance portability
- Problems and their formulations vary in degree of performance gain opportunities through dynamic methods
- Runtime system software is a near-term strategy to enhance parallel system performance through overhead reduction, minimizing latency effects, avoiding contention, and exploitation of increased parallelism
- Algorithms will need to be refactored in some cases to allow much parallelism
- New interface protocols will be required between compiler and runtime
- OS and runtime relationship can be extended for adaptive control
- Operational semantics can be used for formal specification of execution models to ensure correctness, completeness, and compliance
- Event-driven computation and synchronization can mitigate asynchrony
- Global Address Space semantics yield improved programmability but impose additional overhead implementation challenges
- Advanced methods of introspection and policies require further advances

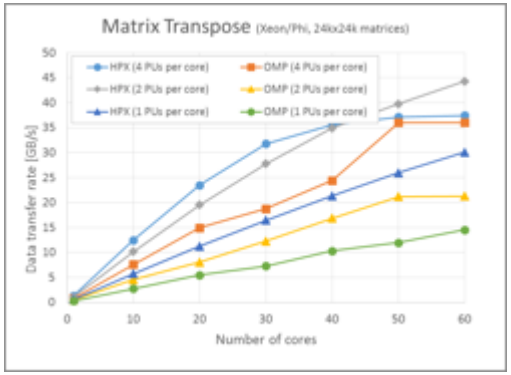
HPX-3

Higher Level APIs for Portable Performance (HPX-3)

- Problem
  - Different Architectures often require separate optimization and codes
- Solution
  - Offer a higher level C++ programming interface for HPX that is well aligned with the modern C++ standard and which enables full portability of code and performance
  - Uniform code for GPUs and main codes
- Recent results
  - Parallel APIs implemented in HPX have been adopted for the next C++17 standard
  - A newly written (local and distributed) HPX matrix transposition benchmark outperforms a similar code from the Intel Parallel Research Kernels (based on OpenMP and MPI) by a large margin (up to 50% faster)
  - The new benchmark shows excellent performance on different architectures (Intel X86, Xeon/Phi, and GPUs)
- Impact
  - Application developers can write new code once using a higher level API and efficiently run it on many platforms







Future Adoption of Higher Level APIs (HPX-3)

- Evaluation Plans
  - Measure User base
    - Current projects (STAR, PXFS, STORM, Parquet)
    - Use in global projects, like H2020, Human-Brain project, CERN
  - Measure impact on open source community
    - Google Summer of Code
    - Boost
  - Measure impact on standardization efforts (C++17)
  - Interaction with industry (Intel, NVidia, AMD)

- Technology Transition Opportunities
  - Provide implementation and usage experiences for ongoing and future C++ standardization (labs heavily depend on C++)
    - HPX provide uniform local and remote parallelism APIs (parallel algorithms and data structures, GPU integration)
    - HPX integrates high scalability runtimes with existing C++ application infrastructures
  - Work with international communities and companies

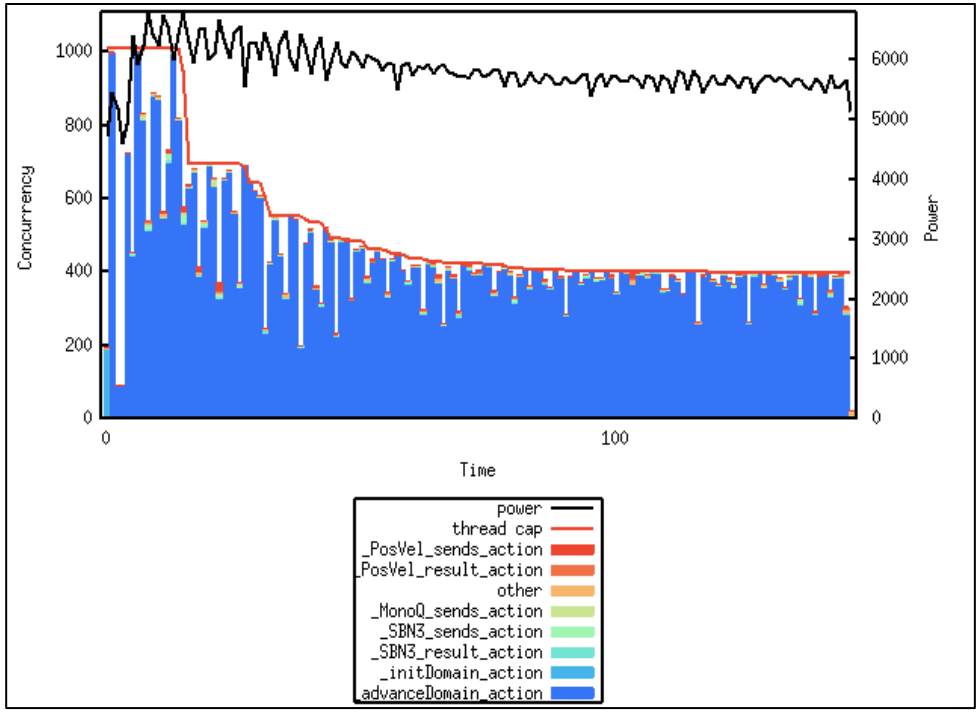
Future Adoption of Higher Level APIs (HPX-3)

- Lessons learned
  - Task based parallelism can provide a most efficient technological bases for any kind of higher level parallelism constructs in C++
  - Higher level APIs in C++
    - Simplify writing application code
    - Outperform existing programming models (OpenMP, MPI, CUDA)
    - Ensure portability of code and performance across heterogeneous platforms
  - Runtime adaptivity is key for efficient applications
    - Gives emergent properties supporting high scalability

DEMOS

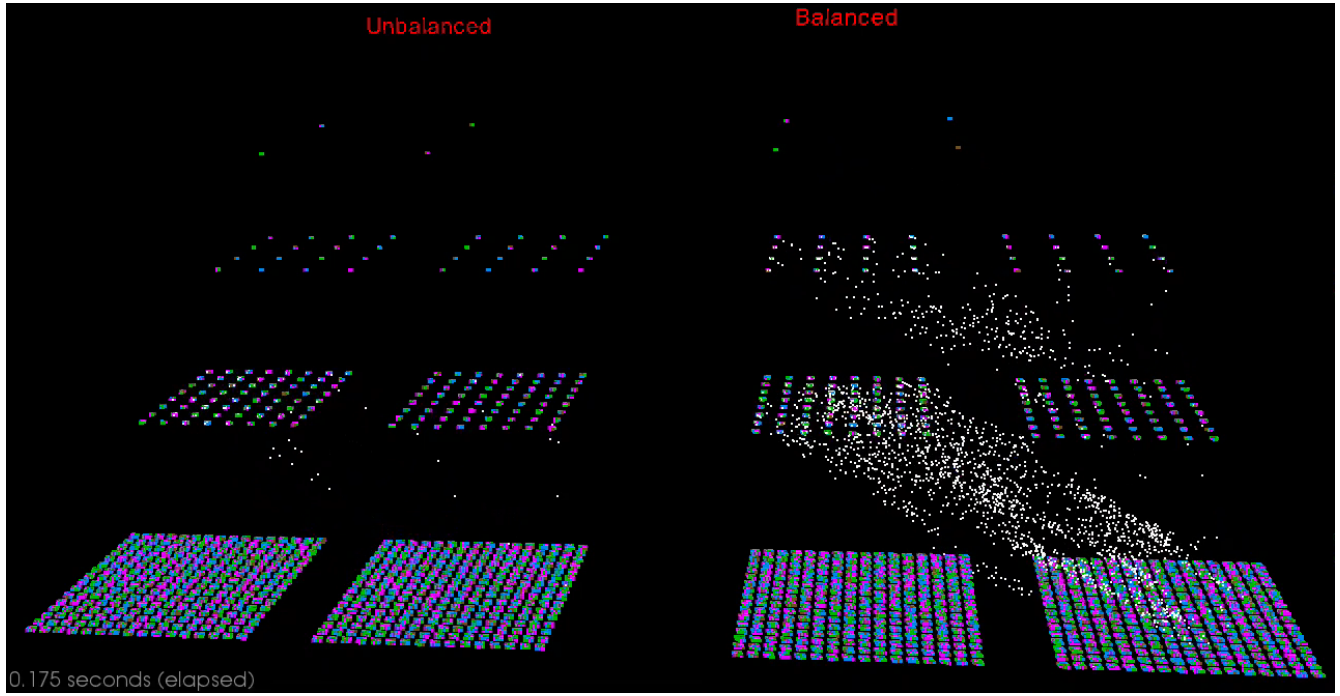
Technology Marketplace Demos - Demo 1

Live demo shows the performance scalability of HPX-5 integrated with the Autonomic Performance Environment for Exascale (APEX) running LULESH application



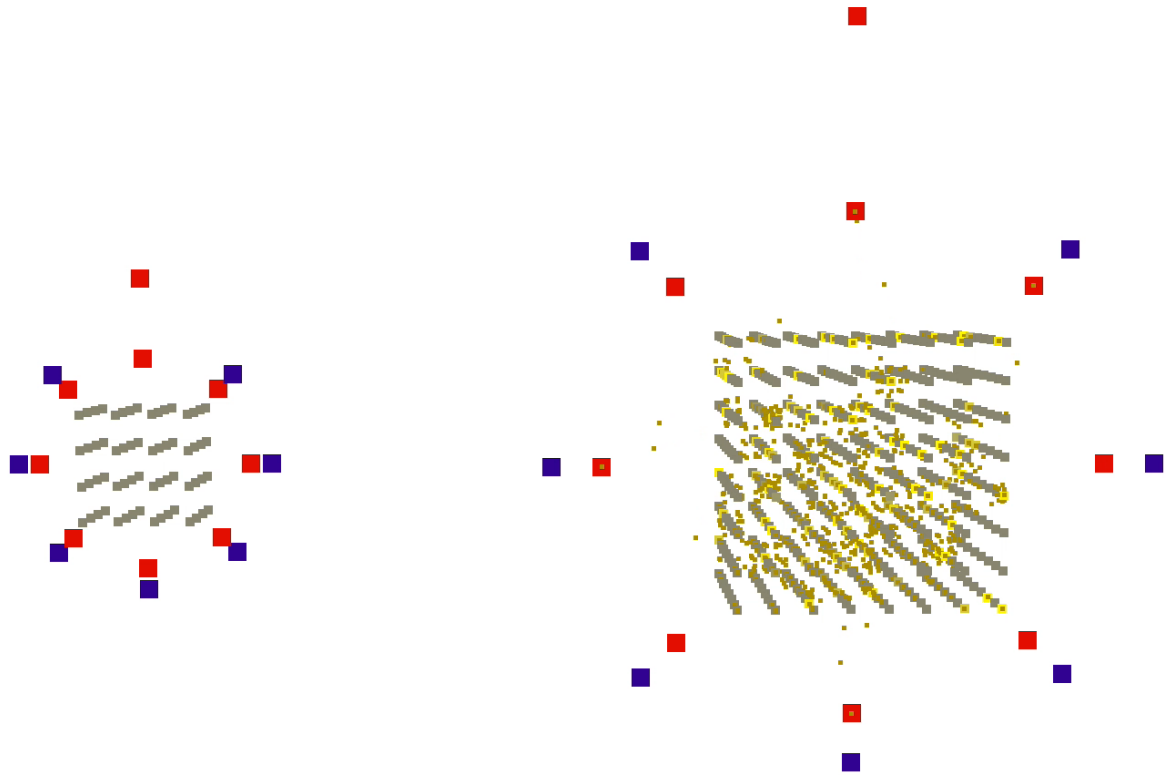
Demo2 - Dynamic Adaptive Nature of the Runtime

Using the fast multipole method (FMM) application, this visualization shows the difference between remote communication activity before and after dynamic rebalancing effected by the active global address space (AGAS) in HPX-5.



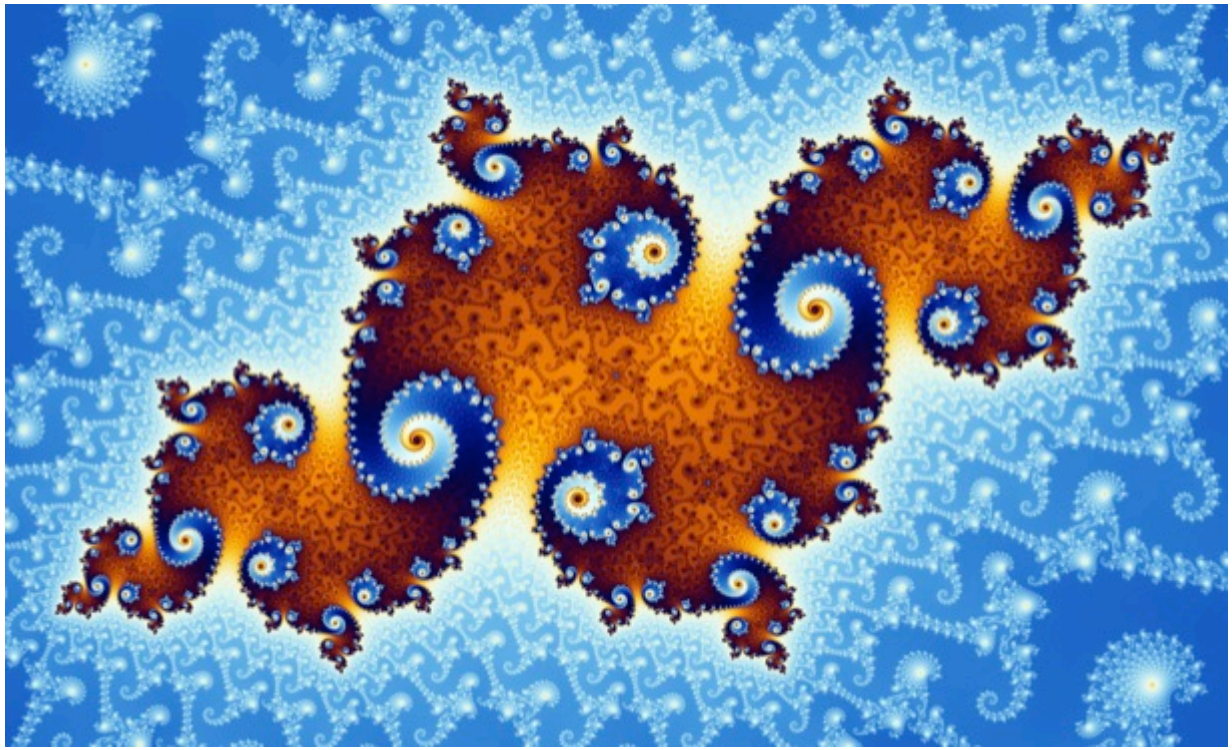
Demo2 – LULESH Visualization

This visualization shows the benefit of asynchronous behavior from the over-decomposition in HPX-5 for LULESH application



Demo3 – Demonstration of HPXCL

Demonstrates HPXCL, a scalable OpenCL API for distributed systems, on top of LSU's HPX-3 (a scalable C++ runtime system), with distributed Mandelbrot renderer



Demo 4 – Xstack Integration demo

LULESH application running on KNL Pre-release hardware with the entire integrated software stack (LXK, APEX, RCR and HPX-5)

ktpedre — ktpedre@gato2:~ — ktpedre@gato2:~ — ssh — 125×35

```
In __task_create(), starting task 202 on cpu_id=201
In __task_create(), starting task 203 on cpu_id=202
In __task_create(), starting task 204 on cpu_id=203
In __task_create(), starting task 205 on cpu_id=204
In __task_create(), starting task 206 on cpu_id=205
In __task_create(), starting task 207 on cpu_id=206
In __task_create(), starting task 208 on cpu_id=207
In __task_create(), starting task 209 on cpu_id=208
In __task_create(), starting task 210 on cpu_id=209
In __task_create(), starting task 211 on cpu_id=210
In __task_create(), starting task 212 on cpu_id=211
In __task_create(), starting task 213 on cpu_id=212
In __task_create(), starting task 214 on cpu_id=213
In __task_create(), starting task 215 on cpu_id=214
In __task_create(), starting task 216 on cpu_id=215
In __task_create(), starting task 217 on cpu_id=216
In __task_create(), starting task 218 on cpu_id=217
In __task_create(), starting task 219 on cpu_id=218
In __task_create(), starting task 220 on cpu_id=219
In __task_create(), starting task 221 on cpu_id=220
In __task_create(), starting task 222 on cpu_id=221
In __task_create(), starting task 223 on cpu_id=222
In __task_create(), starting task 224 on cpu_id=223
In __task_create(), starting task 225 on cpu_id=224
In __task_create(), starting task 226 on cpu_id=225
In __task_create(), starting task 227 on cpu_id=226
In __task_create(), starting task 228 on cpu_id=227
In __task_create(), starting task 229 on cpu_id=228
In __task_create(), starting task 230 on cpu_id=229
In __task_create(), starting task 231 on cpu_id=230
In __task_create(), starting task 232 on cpu_id=231
In __task_create(), starting task 233 on cpu_id=232
<8>(init_task)  Number of domains: 64 nx: 15 maxcycles: 400 core-major ordering: 1
<8>(init_task.thread_03) power : 0.782609, ma: 0.518519, cap: 230, min: 0.695652, max: 1.000000, no change.
```

http://xstack.sandia.gov/xpress