XPRESS

e**X**ascale **PR**ogramming **E**nvironment and **S**ystem **S**oftware

Preliminary Progress Report: Year 1
September 1, 2012 -- March 15, 2013

The XPRESS project is a collaborative effort across eight institutions that include laboratories and universities. Overall leadership for the integrated project is provided by Sandia National Laboratories. Other laboratories involved are Oak Ridge National Laboratory and Lawrence Berkeley National Laboratory. University partners include Indiana University, Louisiana State University, the University of Oregon, the University of North Carolina at Chapel Hill, and the University of Houston.

Contents

Goal

The goal of the XPRESS project is the research and development of OpenX, a complete software architecture for Exascale computing. Four major R&D thrusts include:

1) an Exascale lightweight kernel operating system (LXK), based on the Kitten OS, to manage billionway hardware parallelism, management of faults and power, management of global virtual name space, and other features of future system architectures;
2) a runtime system (HPX-4), co-designed with LXK, which will be based on the ParalleX execution model and will support dynamic resource management and task scheduling;
3) system interfaces for interoperability between the runtime system and both the OS and APIs; and;
4) compilation strategies and systems to translate MPI and OpenMP legacy codes to a form that can be run by OpenX with performance at least as good as a native code implementation.

Objective

XPRESS will define a system software architecture, OpenX, to represent the full functionality ultimately anticipated for an Exascale computing system. While a paper specification, it will include important interfaces between the programming system and underlying runtime and OS called XPI and between the runtime system and operating system called RIOS. Compliance with these interface specifications will facilitate different X-stack configurations comprising components of different design, possibly by different development teams to accelerate progress towards advanced DOE systems. XPRESS will implement a critical subset of OpenX software modules to integrate and test key functionality to demonstrate and apply a working software system incorporating the defining innovative concepts upon which XPRESS is defined. These will include the LXK lightweight kernel OS and the HPX-4 runtime system interoperating through RIOS and driven by workloads via the XPI interface.

Technical Strategy

XPRESS is organized as a set of cooperative tasks to develop and test OpenX, a software architecture based on the concept of delivering working scalable and efficient runtime environments for Exascale computing. These major tasks include:

- OpenX software architecture – a conceptual framework for the co-design and interoperability of proof-of-concept XPRESS software stack

- ParalleX execution model – guiding principles for co-design of components of the OpenX stack with advances in locality management and task prioritization through introspection

- HPX runtime system – support of application dynamic adaptive resource management, task scheduling, and introspective control policies

- LXK operating system – lightweight kernel operating system for order-constant scalability and low/no noise to manage resources

- RIOS – a realm of Exascale system operation unique in the X-stack program. The full system stack including the relationship between the new generation of lightweight kernel operating systems and runtime system software

- XPI advanced programming model – intermediate form and low-level (readable) programming interface reflecting the ParalleX model, providing a target for source-to-source high level parallel language translation, and supporting early direct programming experimentation and measurement

- Performance models & metrics – provide parameters and their mutual sensitivities to guide co-design and quantify operational behavior

- Legacy application mitigation – ensuring seamless transition of legacy codes and programming methods to the future generation of ParalleX-based Exascale systems

- Experiments and evaluation – critical to determining degree of effectiveness and likelihood of ultimate success as well as guiding corrective design changes to achieve DOE objectives

- Applications – collaborations with Co-Design Centers and other mission critical codes

- Documentation – as well as reporting to DOE X-stack program management, to provide early adopters with sufficient information to apply prototype programming and execution environment

Schedule and Milestones – Year 1

The schedule for year 1 of the XPRESS project includes the following major milestones and tasks related to the development of OpenX:


Indicates that the task is progressing as planned and will be completed on time.


Indicates that the task is progressing slower than planned, and there may be delays.


Indicates that the task has not been started.

Component	Year 1 Tasks	Completion Date	Status	Progress	Lead Institution(s)
OpenX Software Architecture	• Define architecture components	8/31/2013		• Defined the components of the for Exascale computing, including ParalleX execution model, LXK operating system, HPX-4 runtime system, interface protocol, compilation methods, debugging tools, instrumentation, fault tolerance, and power management	SNL, IU
ParalleX Execution Model	• Refine specification - extended to incorporate the semantics of locality and priority policies	8/31/2013		• Devised an initial simple model of locality t distinguishes among pair-wise associations with respect to their relative locality	IU
HPX-3	• Implement processes, object migration,	8/31/2013		• Released HPX V0.9.5 (API and	LSU

	policies, system introspection • Develop, maintain, document, and support HPX-3 as a bootstrapping platform for newly designed XPI API			performance counter improvements) • Released HPXC V0.2 (pthreads)	
HPX-4 Runtime System	• Development of the software architecture for each of the component subsystems	8/31/2013		• Developed the top-level software architecture for the HPX-4 parcel handler and synchronization	IU
LXK Operating System	• Port HPX-3 on Kitten	8/31/2013		• Completed an initial analysis of the operating system (OS) requirements for the HPX-3 runtime system • Re-enabled support in Kitten for Mellanox InfiniBand (IB) network devices • Added support for Qlogic IB cards to allow Kitten to run on Cutter at Indiana • Added support for using the Portals 4 network programming interface to Kitten for intra-node inter-process communication • Completed a preliminary analysis of the functionality needed to support network-based task spawning through Portals	SNL, LSU
RIOS	• Develop first draft protocol of the RIOS interface	8/31/2013		• Explored the parcel interface which supports message-driven computation to be conducted across the system and between nodes • An initial draft of this interface will be available by the end of year one	SNL, IU
XPI	• Develop first specification	8/31/2013		• Developed first full specification of the XPI programming interface	IU
Compilation/ Introspection	• Design and prototype HPX-LXK interface for performance information • Design methodology and development approach for performance introspection in the HPX runtime • Help design XPI interface to allow for maximal system performance and parallelization information transfer	8/31/2013		• Designed how the information flow between the initial performance tools will occur and have • Started the implementation to merge the tools and performance models. • Improved the collection of information about dynamic system performance • Re-implemented the prototype performance daemon to reduce the overhead by 10-20x	UNC/RENCI
APEX Performance Measurement	• Design methodology and development approach for APEX performance instrumentation and measurement integration with OS and runtime layers • Develop initial version of measurement wrapper libraries for XPI • Implement performance observation support in HPX-3 and evaluate • Identify HPX-4 performance requirements based on HPX-3 observations	8/31/2013		• ParalleX model studied, requirements for APEX measurement informally captured, APEX prototype implemented • Awaiting XPI Draft 1 • HPX-3 instrumented with APEX measurement • Monitoring HPX-4 design to provide APEX support when implemented	UO
Legacy Migration	• Baseline runtime on HPX-3 on kitten • Explore baseline support to port OpenMP/OpenACC codes to XPI • Evaluate the modifications required to support Open MPI to OpenX	8/31/2013		• Defined a strategy for adapting legacy MPI and OpenMP programming models to HPX based on the OpenUH OpenMP runtime and the OpenMPI MPI library • Started development of the OpenACC compiler, which will help the migration of current OpenACC code in the DOE lab to use within HPX runtime • A prototype implementation of data-	UH

				driven OpenMP execution model has also been completed	

Table 1: Year 1 Schedule and milestones.

Activities and Findings – Year 1

The XPRESS research and development activities are helping to define the components of the OpenX software architecture for Exascale computing. Major activities for year 1 include refining the ParalleX execution model, developing the HPX-4 runtime system, LXK operating system, and XPI and RIOS interface protocols, measuring performance, and migrating legacy codes.

ParalleX Execution Model

The ParalleX execution model provides a conceptual framework for the organization of computational elements and their interrelationships. Like other execution models, it defines the guiding principles that govern the form and operation of actions and the data upon which they are performed. In particular, ParalleX unifies a diversity of semantics of parallelism to achieve greater scalability in the era of multicore. It incorporates strategies to manage the uncertainty of asynchrony for high efficiency. It permits the use of runtime software systems to remove the burden of scheduling and resource management from the programmer for greater productivity and performance portability. ParalleX provides a crosscutting abstraction to guide the co-design of the many system layers from programming models to hardware architecture and all of the system software layers in between. It governs their interoperability and mutual support in achieving the desired emergent behavioral properties.

The foundational work in the derivation of the initial ParalleX execution model was conducted under funded research previously sponsored by DOD and NSF. It established a model with basic elements of dynamic multi-threading, parcel message-driven computation, active global address space naming, and local control object synchronization. This basic model has been tested through the experimental development of early runtime system software and its use for new highly scalable applications. The results are promising, showing superior performance in some cases with respect to conventional practices.

ParalleX at the beginning of the XPRESS project had demonstrated the merit of its basic and distinguishing concepts. But it also revealed the need for additional capabilities to permit more intelligent management of resources with respect to the dynamic tasks. Under the XPRESS project, the ParalleX execution model is to be extended to incorporate the semantics of locality and priority policies. The objective of locality management is to balance the need to distribute data and tasks to achieve the highest possible parallel processing for scalability with the need to co-locate data that are used together to minimize latency and overhead for high efficiency. In the current reporting period, an initial simple model of locality has been devised that distinguishes among pair-wise associations with respect to their relative locality. The three classes of pairs are data to data, task to task, and data to task. The types of localities are tightly bound, weakly bound, not bound, and forced separation. This establishes a relative push-pull relationship determining preferential placement of new data and actions or with availability of storage or thread processing resources. This characterization can be bound to placement of the many existing objects to yield an objective function for which optimization will determine ultimate allocation of new objects to distributed resource placement.

HPX-3

LSU's work focuses on the conceptual and design phase of implementing ParalleX concepts in HPX. ParalleX concepts, including ParalleX processes, heterogeneous system support, and distributed AGAS implementation require extensive research to design and implement. ParalleX (PX) parallel processes are special entities that enable forming a hierarchical namespace in the ParalleX execution model. Each PX-process manages a part of the namespace; it participates in address resolution for all its children (any ParalleX first class objects, such as other processes, threads, LCOs, etc.) The objective of PX-processes is to enable the coherent management of computation across localities. Semantically, PX-processes can be viewed as special data structures that hold information pertinent to a task or hold codes/instructions that need to be acted upon along with data. Processes are hierarchical, with the main process at the root of that hierarchy and their children residing at lower levels (see Figure 1). They can protect data from being accessed by any other process. Each PX-process may span multiple localities, i.e. the data managed by a process may be distributed. The set of localities associated with a PX-process may change at runtime as part of the data a PX-process manages may be migrated to other localities.

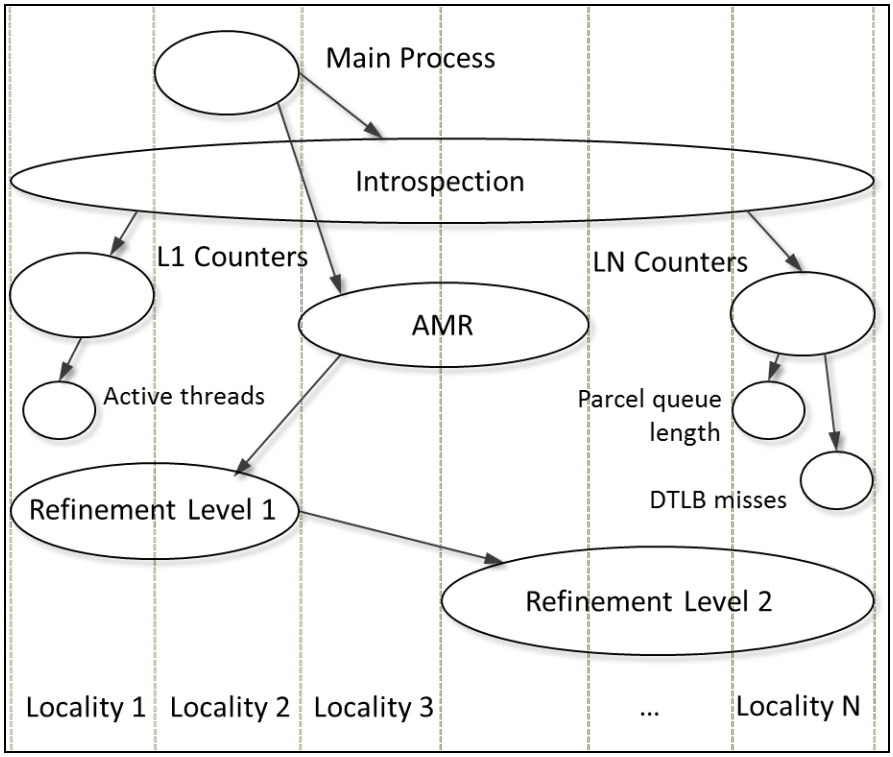


Figure 1: PX-processes rely on distributed data structures, are able to span multiple localities (physical nodes), and this set of localities could be dynamic. Each PX-process supports certain key attributes such as task scheduling policies, access rights interfaces to kernel handlers. Processes are fundamental blocks all tasks/services are built upon. To support these properties, PX-processes maintain members such as thread constructors, synchronization constructs, thread scheduling policies, performance counters, data containers, and others. This figure depicts an example for a set of PX-processes for a hypothetical application, demonstrating their hierarchical and distributed nature. Any of the PX-processes spans a different set of localities and encapsulates a different set of functionalities (code) and related data structures. They expose an individual set of methods representing their encapsulated functionality.

One of the great challenges of parallel programming in conventional systems such as MPI is the semantic gap caused by local virtual memory boundaries. Traditional clusters often consist of multiple

interconnected SMP machines that have no virtual memory infrastructure that spans the entire system. Partitioned Global Address Space (PGAS) systems (such as Co-Array Fortran, Chapel, or UPC) address this semantic gap by implementing a global address space that encompasses multiple compute nodes networked by mainstream interconnects such as Ethernet or Infiniband. PGAS statically partitions logical regions to create this global address space. This model eases parallel programming and is sufficient only for applications that are not highly dynamic and systems that do not add or remove hardware resources at runtime. Moreover, while systems based on PGAS provide support for irregular data structures in general, the Single Program Multiple Data (SPMD) implementations of these systems do not support those. However, global address space solutions such as PGAS are insufficient for the class of applications that are investigated by this project. Parallel applications of highly dynamic nature require equally dynamic systems to manage their resources and ensure optimal parallelization through adaptive load balancing. The Active Global Address Space, a core component of the ParalleX execution model, provides the crucial functionality in support of such a system. AGAS implements a global naming service that permits tracking of physical locations of all first class objects in a parallel computation by providing an efficient translation of object names to resources that host them. As a consequence, AGAS allows these objects to migrate to remote physical resources and supports the addition and removal of localities at runtime.

As a result of the work, LSU identified the following design requirements for AGAS. The development of migration requires AGAS to fulfill certain requirements and semantic guarantees. The primary criteria regard the scalability and correctness of the AGAS services. They are summarized below:

- *Correctness*

  AGAS must make certain guarantees about the accuracy of its address resolution services. Correctness does not mean that AGAS must always give the correct answer; it merely means that the circumstances in which AGAS may answer incorrectly are well defined. There are three fundamental correctness requirements which an AGAS implementation must meet to enable migration: 1) the Parcel Delivery Guarantee, 2) the Reference Validity Guarantee, and 3) Correctness While Determining Proximity.

- *Distributed AGAS Services*

  Service of address resolution becomes a system bottleneck when a single AGAS server must handle potentially hundreds of thousands of concurrent address resolution requests. While a single AGAS server eases the enforcement of the aforementioned correctness guarantees, experience shows it becomes an unacceptable overhead in a system with hundreds of thousands of localities (not uncommon in present-day systems). The solution to this bottleneck is for the AGAS server to be distributed across multiple localities.

  The goal is to create such a distributed AGAS service in the context of XPRESS. This AGAS service will be closely related to PX-processes. Each PX-process spans part of the namespace, thus allowing it to resolve all managed references. For this reason each instance of a PX-process is a part of the AGAS service and is responsible for resolving the names of all of its children.

LSU also maintains, improves, documents, and supports HPX as the initial platform for XPRESS, and specifically, HPX-3 as a bootstrapping platform for the newly designed XPI API. Legacy work targets MPI

and OpenMP interfaces for the interfaces to the XLK. Two major releases to HPX occurred in the reporting period: HPX V0.9.5 and HPXC V0.2.

- *HPX V0.9.5*

  LSU released HPX V0.9.5 under the Boost Software License V1 (http://www.boost.org/users/license.html), available from Github (https://github.com/STEllAR-GROUP/hpx/). The main goals for this release have been a) API consolidation, b) performance improvements, and c) overall usability improvements.

  The APIs have been aligned with the C++11 Standard, which significantly improves its usability and reduces any learning curve which might have existed before. A strict syntactic and semantic alignment of the HPX APIs with what is mandated by the C++11 Standard and with additional discussion papers which will likely be accepted into the next C++ Standard by 2015 puts HPX in a very strong position in terms of usability, simplicity, and maintainability. This also simplifies the task of documenting those APIs as many resources are already available describing the use of the standard C++11 facilities. Many of the remaining API functions have been adapted for overall consistency.

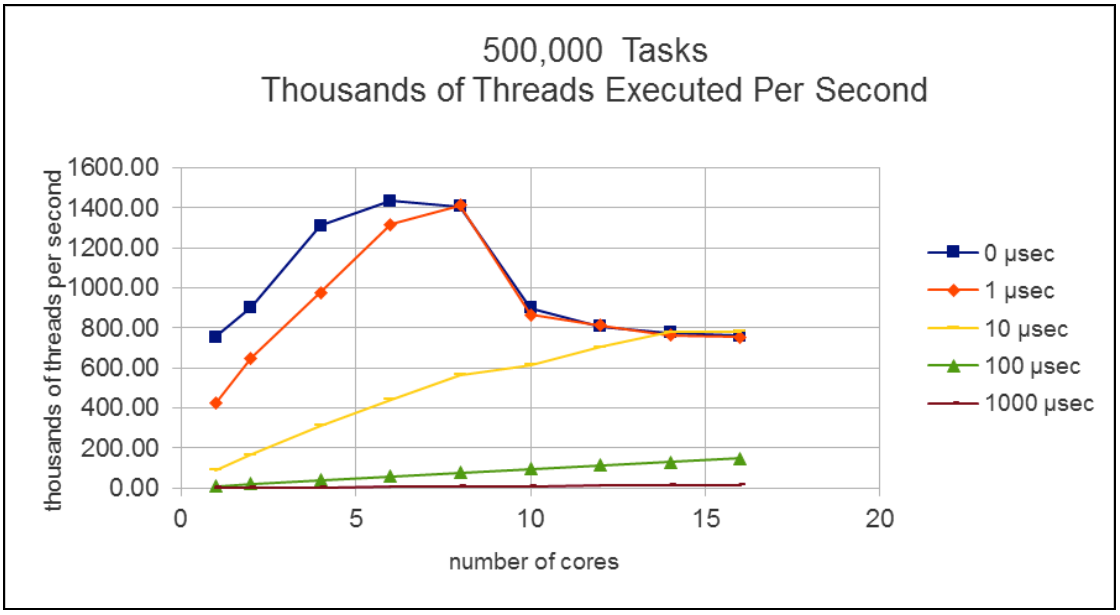


Figure 2: Number of executed HPX threads over the number of cores utilized in a SMP system for different amounts of (artificial) workload

The API now provides asynchronous functions wherever the execution of the API function could result in remote operations or could take more than ~100µs to execute. Using those API functions has the potential to improve the overall application performance by allowing overlap computation with communication and enables fine grain parallelism and synchronization.

This work resulted in greatly improved performance of the threading and parcel transport subsystem layers of HPX. The (amortized) overheads for creating, scheduling, executing, and deleting one HPX thread (with no workload) were reduced to 700ns, which is equivalent to executing ~1.4 million HPX threads per seconds (see Figure 2).

The performance counter framework, which is a special subsystem in HPX, has been improved and refined, and underwent a major redesign aimed at higher performance and greater utility. LSU implemented more than 50 new performance counters which expose many of the important HPX system characteristics, such as queue lengths, idle rates, wait times, and amount of data communicated over the network. Many of those changes were inspired by discussions with other groups in the project, mainly related to APEX and RCR-Blackboard, two tools developed by University of Oregon and RENCI for the XPRESS performance subsystem.

HPX documentation has been considerably extended and improved to include a user manual, reference documentation, and many examples. It is available here: http://stellar.cct.lsu.edu/files/hpx_master/docs/html/index.html.

HPX underwent important changes and optimizations in its parcel layer. The overall performance (bandwidth) of the existing TCP/IP based parcel port was improved. A new parcel port enabling shared memory-based parcel delivery between localities running on the same SMP node was implemented. The shared memory parcel port improves the performance of sending parcels significantly. First, measurements indicate improvements in the range of 50% better bandwidth compared to the existing (now optimized) parcel port based on standard TCP/IP sockets. Work on implementing a parcel port that directly binds to the RDMA InfiniBand interface is started. This work is not complete at the time of this writing.

- *HPXCV0.2: A pthreads (compilation) compatibility later*

HPX provides cutting-edge facilities for parallel programming including a high-performance user thread task manager and an API which supports distributed futures, data flows, and remote objects. Fully taking advantage of the advanced API often requires significant reworking of an existing code. However, by providing a more familiar alternative interface, it should be possible for HPX to facilitate increased concurrency for legacy applications through its support for massive numbers of threads.

Toward that end, LSU has created HPXC, a lightweight interface to HPX with C-language bindings that resembles the pthread library. This effort does not intend to replicate the entire set of pthread functions, instead focusing on the most popular and significant pthread functions. Transforming a code to use HPXC is straightforward, requiring only trivial edits or inclusion of a header file:

```
#include <pthread_to_hpxc.h>
```

This header redefines all pthread symbols to their HPXC equivalents. **Error! Reference source not found.** shows a list of pthread functions supported by HPXC together with notes on important considerations for codes wishing to switch from pthreads to HPXC.

HPX-4

The HPX-4 runtime system will be a complete rewrite of HPX to provide a new generation of dynamic adaptive execution capable of serving POSIX-based HPC systems as well as the new LXK lightweight kernel Exascale operating system. HPX-4 has four principal components: thread manager, parcel handler, local control object synchronization, and global address space server. LSU is responsible for

delivering the new thread manager. IU is contributing the parcel handler and synchronization. The first task towards the realization of HPX-4 is the development of the software architecture for each of the component subsystems. IU has developed the top-level software architecture for the HPX-4 parcel handler and synchronization. It comprises interfaces, queue structures, runtime services (RTS), and the action dispatch function. The parcel manager interfaces with the system area networks, the operating system, the thread manager and global address space server, and the main memory system including the DMA controller. The RTS functions include: a) parcel receive, b) parcel dispatch, c) parcel transmit, d) thread update, e) thread create, f) process create, g) block data movement, and h) AMO execute. There are also RTS functions for LCO event and LCO create in support of global synchronization. A report has been developed documenting this software architecture for HPX-4.

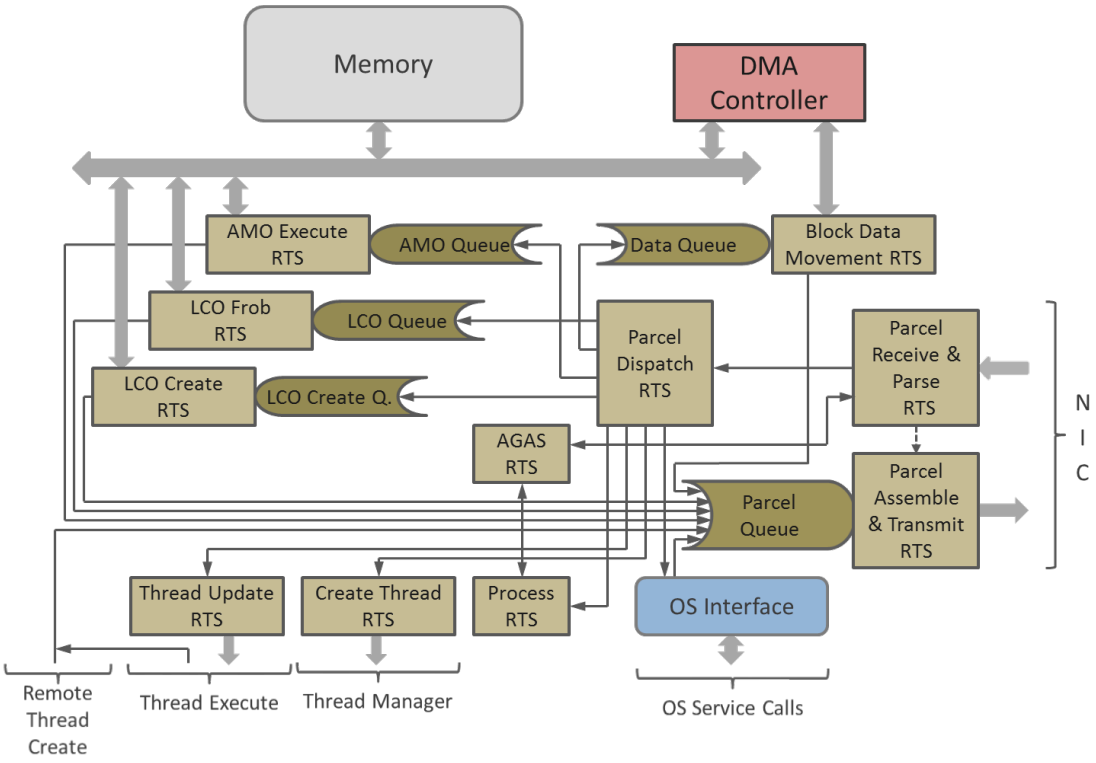


Figure 3: HPX-4 Parcel Handler

LXK – Lightweight OS

SNL completed an initial analysis of the operating system (OS) requirements for the HPX-3 runtime system. This analysis revealed several capabilities that were not available in the Kitten lightweight kernel, including dynamic library support through the `dlopen()` interface, handling of basic standard I/O system calls, and intra-node inter-process communication facilities. These capabilities are now under development and we expect to complete them in time to meet the deliverable of running HPX-3 on Kitten at the end of the first year. During this time, SNL also re-enabled support in Kitten for Mellanox InfiniBand (IB) network devices. Previous versions of the Open Fabrics device driver stack needed for IB worked with Kitten, and changes were needed to get the latest driver code working. Support was added for Qlogic IB cards to allow Kitten to run on Cutter at Indiana. In order to enable scalable process launch, SNL is integrating support for the Process Management Interface (PMI) into Kitten. PMI is a standard interface that many existing libraries, including MPICH and OpenMPI, can use

to start remote processes on nodes within a cluster. SNL is integrating into Kitten the functionality needed by the Hydra process launch system that uses PMI and expect to complete that work within year one. SNL has also added support for using the Portals 4 network programming interface to Kitten for intra-node inter-process communication. SNL is developing an implementation of the Parcels communication layer on top of Portals 4 and have added support to Kitten for ensuring message progress for transfers between separate address spaces on the same node. SNL has also done a preliminary analysis of the functionality needed to support network-based task spawning through Portals.

RIOS

XPRESS is exploring a realm of Exascale system operation unique in the X-stack program: the full system stack including the relationship between the new generation of lightweight kernel operating systems and runtime system software. The "Runtime Interface to Operating System" or "RIOS" is a major component of the overall OpenX Exascale software system architecture. It codifies the interrelationships between the runtime and OS, the data exchange and events that must be communicated between the two layers, and the requirements of the protocol that must ultimately be specified. Towards this end, IU has focused on a key aspect of the RIOS: the parcel interface, which supports message-driven computation to be conducted across the system and between nodes. Parcels will usually come from other nodes but may also be used as a protocol of communications between the OS and the runtime system even on the same node. This is helping to define the emerging protocol.

Development of the first draft of the RIOS interface is in progress. Several project meetings at Sandia have been conducted to analyze the various resource and management functionality required to support dynamic adaptive runtime systems such as HPX. An initial draft of this interface will be available by the end of year one.

XPI – Intermediate Form for Program Parallel Representation

XPI (eXtreme scale Programming Interface) is an intermediate form of abstraction for advanced parallel execution. It is a syntactical representation of the ParalleX execution model and a stable interface layer to the HPX runtime system. It serves as a target for source-to-source compilation from high-level programming languages and compilers. It also provides a low-level, user-readable syntax for direct programming. As such, it provides an early formalism for application-driven experimentation and parallel algorithm development. It will also make possible a common framework to unify access to high level programming models like DSLs and co-design applications that are captured in this form for the X-stack Program and other related initiatives.

The XPRESS XPI task has developed its first full (and mostly complete) specification of the XPI programming interface. It is based on the library model of implementation with bindings to the C programming language. Categories of defined instructions include: 1) miscellaneous, 2) threads, 3) parcels, 4) processes, and 5) AGAS. The syntax of commands exhibits an appearance very similar to that used for MPI. The data types used for parcel data payload transfer are derived from those of MPI including compound data types. Of course the semantics are markedly different. While the model is intrinsically dynamic, options are being included that will permit programmers to make static many of the resource allocations and task assignments. A report of the current version of the XPI specification has been written and delivered.

Introspection/Compilation

LSU, UO, and UNC/RENCI have designed how the information flow between the initial performance tools will occur and have started the implementation to merge the tools and performance models. Several phases for the integration of information between the tools were identified and work has begun.

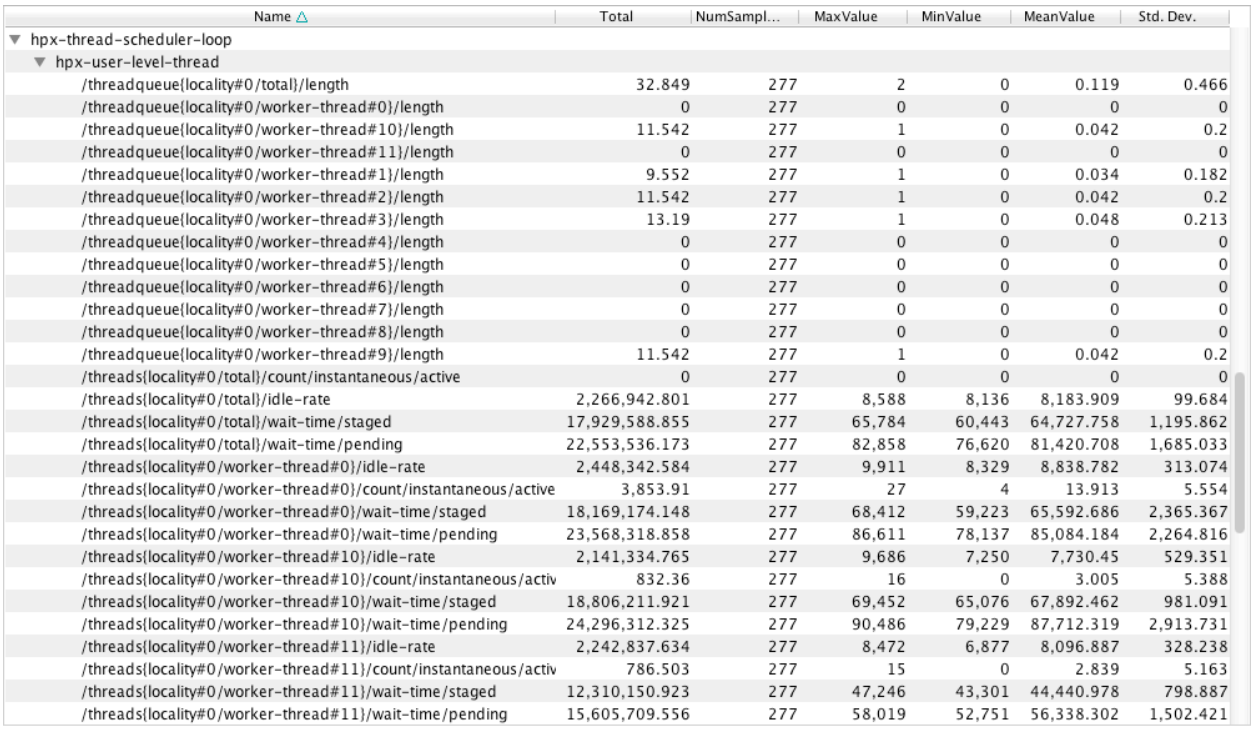
UNC has focused on improving the collection of information about dynamic system performance. The performance collection daemon has been ported to the Intel SandyBridge architecture and expanded to take advantage of the new performance counters (particularly the RAPL power interface). UNC has re-implemented the prototype performance daemon to reduce the overhead by 10-20x. This will allow the daemon to be run in production environments.

APEX Performance Measurement

The XPRESS team has made solid progress with respect to design of the APEX performance measurement infrastructure, performance measurement of the HPX-3 runtime system, and beginning the transition from first-person performance reporting to third-person performance observation. The XPRESS team has implemented an APEX prototype using TAU[1] as the core measurement infrastructure. The APEX instrumentation interface provides access to TAU performance timers and counters. The APEX prototype also works with TAU event-based sampling, providing performance observation of the HPX-3 runtime and application code without instrumentation, which will guide the placement of additional APEX timers in the HPX-3 runtime.

HPX-3 previously provided support for various performance counters that could be periodically sampled and/or reported at program termination. Some examples of these counters include the runtime thread queue length, various thread counts, and the thread idle rate. As these counters were output only to the user terminal, the counters were not easily machine-readable, nor in a form that could leverage existing analysis tools. HPX-3 was modified so that when the counters were observed they were also recorded by the APEX prototype, and subsequently stored in performance profiles at program termination. This initial implementation will be improved, so that whenever these monitored counters are modified, they will be automatically reported to the APEX measurement library, rather than only when requested. Supporting the HPX-3 counters required a modification to TAU to create a new type of counter, a *context user event*. Sample output from a group of HPX-3 counters collected during a run of GTCX is shown in Figure 4. In this example, the HPX-3 counters were observed every four seconds during the run, resulting in 277 total samples. The measurement library also generates simple statistics (maximum, minimum, mean, and variance).

[1] http://tau.uoregon.edu



Name	Total	NumSampl...	MaxValue	MinValue	MeanValue	Std. Dev.
▼ hpx-thread-scheduler-loop						
▼ hpx-user-level-thread						
/threadqueue{locality#0/total}/length	32.849	277	2	0	0.119	0.466
/threadqueue{locality#0/worker-thread#0}/length	0	277	0	0	0	0
/threadqueue{locality#0/worker-thread#10}/length	11.542	277	1	0	0.042	0.2
/threadqueue{locality#0/worker-thread#11}/length	0	277	0	0	0	0
/threadqueue{locality#0/worker-thread#1}/length	9.552	277	1	0	0.034	0.182
/threadqueue{locality#0/worker-thread#2}/length	11.542	277	1	0	0.042	0.2
/threadqueue{locality#0/worker-thread#3}/length	13.19	277	1	0	0.048	0.213
/threadqueue{locality#0/worker-thread#4}/length	0	277	0	0	0	0
/threadqueue{locality#0/worker-thread#5}/length	0	277	0	0	0	0
/threadqueue{locality#0/worker-thread#6}/length	0	277	0	0	0	0
/threadqueue{locality#0/worker-thread#7}/length	0	277	0	0	0	0
/threadqueue{locality#0/worker-thread#8}/length	0	277	0	0	0	0
/threadqueue{locality#0/worker-thread#9}/length	11.542	277	1	0	0.042	0.2
/threads{locality#0/total}/count/instantaneous/active	0	277	0	0	0	0
/threads{locality#0/total}/idle-rate	2,266,942.801	277	8,588	8,136	8,183.909	99.684
/threads{locality#0/total}/wait-time/staged	17,929,588.855	277	65,784	60,443	64,727.758	1,195.862
/threads{locality#0/total}/wait-time/pending	22,553,536.173	277	82,858	76,620	81,420.708	1,685.033
/threads{locality#0/worker-thread#0}/idle-rate	2,448,342.584	277	9,911	8,329	8,838.782	313.074
/threads{locality#0/worker-thread#0}/count/instantaneous/active	3,853.91	277	27	4	13.913	5.554
/threads{locality#0/worker-thread#0}/wait-time/staged	18,169,174.148	277	68,412	59,223	65,592.686	2,365.367
/threads{locality#0/worker-thread#0}/wait-time/pending	23,568,318.858	277	86,611	78,137	85,084.184	2,264.816
/threads{locality#0/worker-thread#10}/idle-rate	2,141,334.765	277	9,686	7,250	7,730.45	529.351
/threads{locality#0/worker-thread#10}/count/instantaneous/activ	832.36	277	16	0	3.005	5.388
/threads{locality#0/worker-thread#10}/wait-time/staged	18,806,211.921	277	69,452	65,076	67,892.462	981.091
/threads{locality#0/worker-thread#10}/wait-time/pending	24,296,312.325	277	90,486	79,229	87,712.319	2,913.731
/threads{locality#0/worker-thread#11}/idle-rate	2,242,837.634	277	8,472	6,877	8,096.887	328.238
/threads{locality#0/worker-thread#11}/count/instantaneous/activ	786.503	277	15	0	2.839	5.163
/threads{locality#0/worker-thread#11}/wait-time/staged	12,310,150.923	277	47,246	43,301	44,440.978	798.887
/threads{locality#0/worker-thread#11}/wait-time/pending	15,605,709.556	277	58,019	52,751	56,338.302	1,502.421

Figure 4: HPX-3 performance counters captured by APEX. The counters are saved in a parallel profile data file, and visualized with ParaProf, the profile visualization and analysis tool.

The HPX-3 runtime thread manager was instrumented with APEX timers. This instrumentation provide insight into how much time is spent in runtime thread scheduling as opposed to actual application processing. An example of a profile collected with these timers is shown in Figure 5. This example was run with 48 total HPX threads on four nodes of ACISS[2], the computational cluster at the University of Oregon. In addition to the 12 OS threads that are directly involved in application execution progress, HPX-3 has eight additional OS threads. Instrumenting HPX-3 in this way will help HPX developers in reducing the runtime thread scheduling overhead.

Figure 6 shows a profile of the same application, but with sampling enabled to gain insight into where the application and runtime are spending the most time in execution. Figure 7 shows a trace timeline of the thread scheduler in HPX-3 when executing the GTCX application.

[2] http://aciss.uoregon.edu

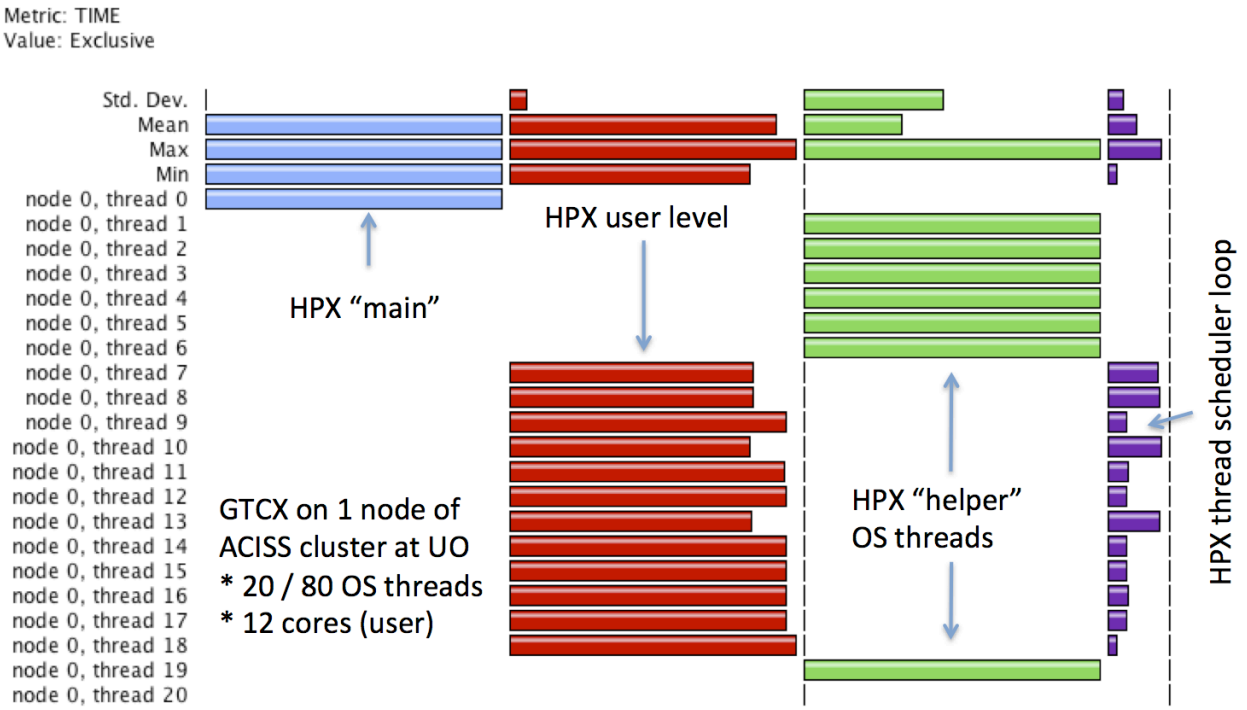


Figure 5: APEX profile measurement of HPX-3 runtime thread scheduler, as visualized in ParaProf. Only the first of four processes is shown. In addition to the twelve worker threads, HPX-3 also has a main thread (blue) and seven helper OS threads (green). The twelve worker threads split their time between the thread scheduler loop (purplered) and user level work (redpurple).

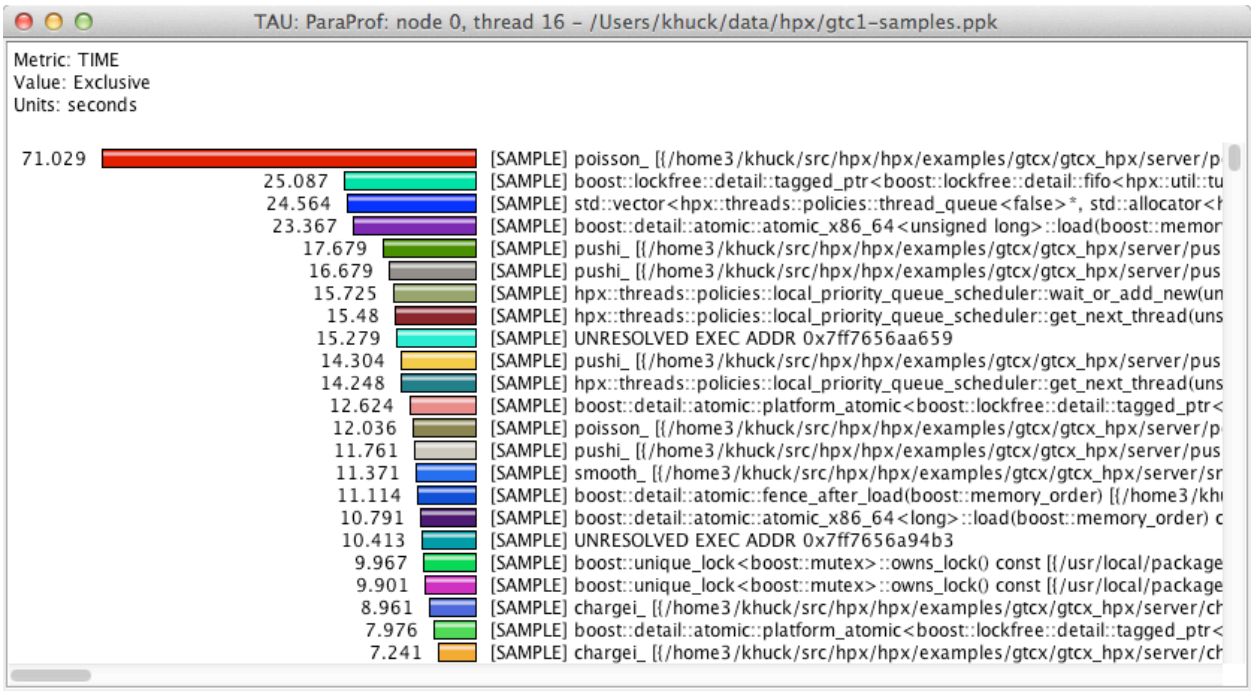


Figure 6: APEX profile measurement of HPX-3 application GTCX, using sampling. The total runtime of the application was 1132 seconds when executed with 12 HPX-3 threads on one node. Most sampled time is spent in

application code (poisson, pushi, smooth), but there is also significant time spent in overhead (HPX-3 code, library calls, other unresolved addresses).

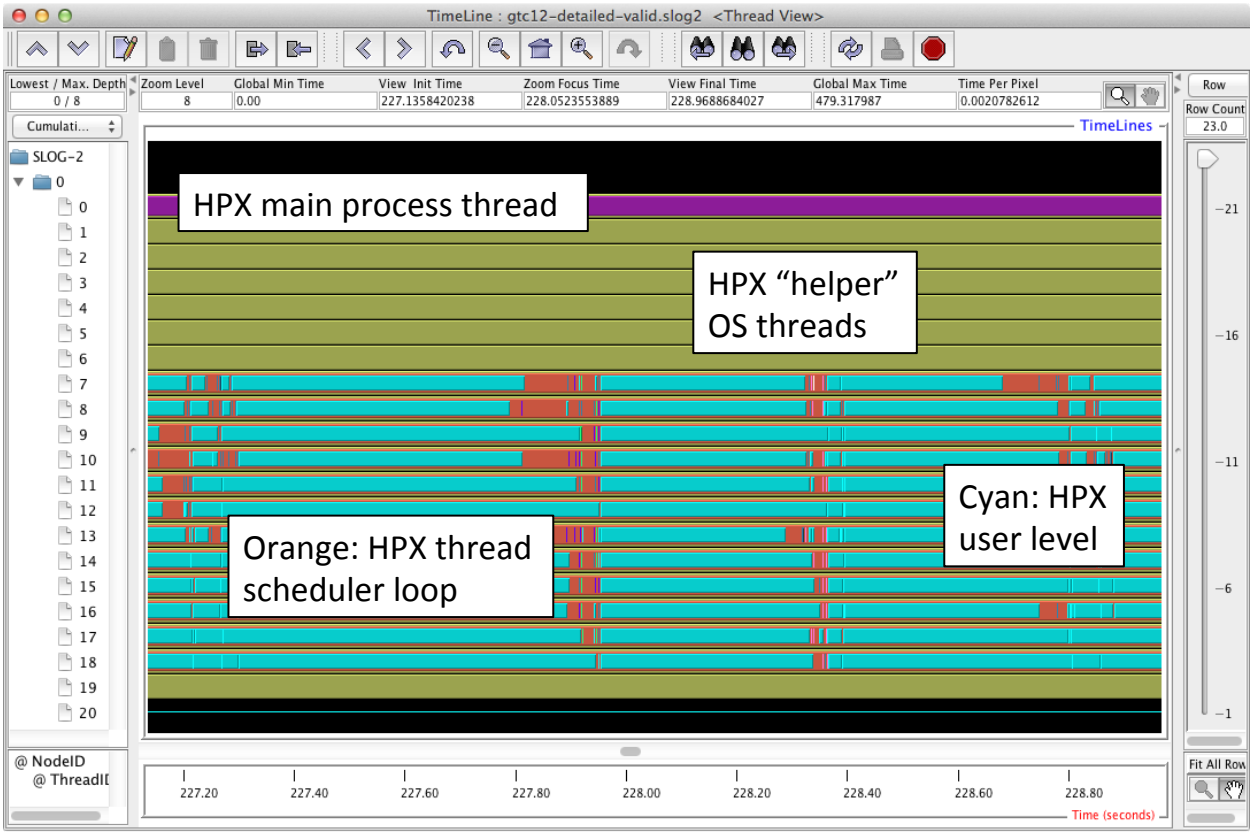


Figure 7: Timeline view of HPX-3 application in Jumpshot, a trace visualization tool. The twelve worker threads are clearly visible (in blue and orange), and alternate their time between the thread scheduler (orange) and work tasks (blue).

The APEX prototype has been developed with the HPX-3 build process in mind, and is designed to integrate seamlessly into the HPX-3 build. If the APEX variables are not defined, HPX-3 is configured and compiled as normal. If the APEX variables are defined with a reference to the local APEX installation, the HPX-3 build system will enable APEX support.

One of the key aspects of the APEX third-person observation design is to incorporate the full system context in any performance observation. With that in mind, the XPRESS team has also been working to integrate the RCRToolkit library into the APEX measurement infrastructure. RCRToolkit exposes holistic hardware and system observations that are not available in the currently executing process space through a shared memory region, called the RCRBlackboard. The APEX prototype has integrated with the RCRBlackboard as a client application, so that non-core hardware measurements such as power and energy can be taken. As the current implementation of RCRToolkit has specific hardware and execution permission requirements, the XPRESS team has established common development systems at both RENCI and LSU that meet the requirements. On the RENCI development system with Intel Sandybridge architecture, APEX has successfully collected non-core hardware measurements such as power and energy from the RCRBlackboard, which is provided by the RCRDaemon running on that system.

While the prototype implementation of APEX provides insight into HPX applications, there are some drawbacks to the prototype approach. The most significant of these is that the performance data is not yet modeled in the ParalleX view of the world. For example, the performance model currently used in APEX uses measurement dimensions that include only the currently executing process and operating system thread, with no explicit support for capturing the HPX-3 runtime thread. However, until the ParalleX performance model is fully captured in APEX there is still valuable insight into the application by implementing the prototype in existing tools, running on existing systems, and mapping back to the ParalleX model.

Legacy Migration

UH, LSU, and IU have defined a strategy for adapting legacy MPI and OpenMP programming models to HPX based on the OpenUH OpenMP runtime and the OpenMPI MPI library. The strategy includes three major steps: 1) retarget OpenMP runtime and MPI to HPX runtime, which will enable seamless migration of MPI/OpenMP codes to use certain features of HPX; 2) add a subset of HPX and XPI to the OpenMP/MPI runtime, as well as appropriate language extensions such that legacy MPI/OpenMP application could use those features mixing with OpenMP/MPI; 3) introduce the full XPI to the OpenMP/MPI runtime and appropriate language extensions to fully exploring the HPX features.

UH has started development of the OpenACC compiler, which will help the migration of current OpenACC code in the DOE lab to use within HPX runtime. A prototype implementation of data-driven OpenMP execution model has also been completed, which will be leveraged with HPX "future" features when integrating the OpenMP runtime with HPX for the step-1 migration goals. Problems envisioned so far are the interoperability issues when integrating the multiple runtime (MPI, OpenMP and HPX), as well as the approach to migrating legacy applications.

Applications

This activity will not start until the second year of the project.

Collaboration w/ Co-Design Centers (and Post Docs)

Thomas Sterling, Chief Scientist of the XPRESS project, is the primary point of contact with the Co-Design Centers regarding applications. He has contacted the following centers to initiate discussion and collaboration:

- John Bell, LBNL, Combustion Exascale Co-Design Center
- Jim Belak, LLNL, Exascale Co-Design Center for Materials in Extreme Environments
- Paul Fischer, ANL, Center for Exascale Simulation of Advanced Reactors (CESAR)

Ron Brightwell, Lead PI of the XPRESS project, has contributed to the "Collaboration and Coordination of the X-Stack Projects with the Co-Design Centers" document.

Project Management

The XPRESS project is led by Ron Brightwell at Sandia National Laboratories. He serves as the coordinating PI. Thomas Sterling, Indiana University, assumes the roles of Chief Scientist and Co-Design Chair for the team. Rebecca Schmitt, Indiana University, is the Project Manager. An Executive

Committee, comprised of lead PIs from collaborating institutions, the Chief Scientist, and Project Manager, provide oversight and execution of the project.

Internal communication mechanisms include biweekly teleconferences and in-person meetings. All team members participate in biweekly teleconferences to discuss technical and administrative updates. Several in-person meetings have also helped facilitate technical progress. Listed below is a timeline of in-person meetings:

- 8/17/2013: Kick-off project meeting at Sandia National Laboratories
- 11/13/2013: Project meeting at Supercomputing 2013
- 1/23/2013: Technical meeting at UH to discuss legacy migration
- 2/6/2013: Technical meeting at LSU to discuss system performance
- 3/7/2013: PI meeting at UNC/RENCI to discuss X-Stack preparations

Documents and code are shared via:

- Email reflectors – xpress@crest.iu.edu, xpress-exec@crest.iu.edu
- Subversion repository - https://svn.osl.iu.edu/rep/xpress/
- Wiki - https://www.crest.iu.edu/projects/xpress/
- XPRESS project website - http://xstack.sandia.gov/xpress/index.html

Education/Outreach/Presentations

Members of the XPRESS team frequently attend and present at meetings, workshops, and conferences related to Exascale computing:

- ExMatEX All-Hands Meeting, 3/12/13, Bernalillo, NM (Sandia)
- Exascale Ecosystem Coordination Meeting, 2/11-13/13, Livermore, CA (Sandia)
- Exascale Research Conference, 10/1-3/12, Arlington, VA (Sandia)
- X-Stack Kickoff Meeting, 9/17-20/12, Portland, OR (Sandia)
- Rice University, 7/27/12, Houston, TX (Sandia)
- Georgia Institute of Technology, 7/31/12, Atlanta, GA (Sandia)

Presentations directly related to XPRESS and Exascale efforts include:

- Lumsdaine, Andrew. "Avalanche: A Flow-Graph Framework for Simplifying the Use of Active Message," Productive Programming Models for Exascale Workshop, 8/2012, Portland, OR. Slides at rlhttp://xsci.pnnl.gov/ppme/pdf/Willcock_pres.pdf
- Sterling, Thomas. "Towards Extreme Scale Computing in the Current Coming Decade," ScalPerf'12, 9/21/12, Bertinoro, Italy.
- Sterling, Thomas. "XPI & RIOS Interfaces to the HPX Runtime System," DOE Exascale Research Conference, 10/1/2012, Arlington, VA
- Sterling, Thomas. "The HPX Runtime System for ParalleX Processing," DOE Exascale Research Conference, 10/1/2012, Arlington, VA
- Sterling, Thomas. "XPRESS: Exascale Programming Environment and System Software," DOE Exascale Ecosystem Coordination Meeting, 10/9-12/12, Oakland, CA

- Sterling, Thomas. “Modeling Execution Models- Top-Down & Bottom-Up,” DOE Exascale Ecosystem Coordination Meeting, 10/9-12/12, Oakland, CA
- Sterling, Thomas. “Exascale HPC Runtime Opportunities and Challenges,” ExaChallenge Symposium, 10/14-20/12, Dublin, Ireland
- Sterling, Thomas. “Broader Engagement and Education in the Exascale Era,” Supercomputing Conference, 11/10-16/12, Salt Lake City, UT
- Sterling, Thomas. “The ParalleX Execution Model for Extreme Scale Computing,” Russian Supercomputing Conference, 11/27-12/1/12, Moscow, Russia
- Sterling, Thomas. “Connections for Coordination of DOE Exascale Research and Development,” 2.11-2.13.13: Exascale Ecosystem Conference, 2/11-13/12, Livermore, CA
- ASCR Summit, 2/28/13, Germantown, MD
- PENDING: 9th Workshop on High-Performance, Power-Aware Computing (in conjunction with IPDPS 13), 5/13/2013, Boston, MA (UNC/RENCI)

Publications

Publications resulting from the XPRESS project include:

- Support For Dependency Driven Executions Among. OpenMP Tasks. Priyanka Ghosh, Yonghong Yan, and Barbara Chapman. The 2nd Workshop on Data-Flow Execution Models for Extreme Scale Computing (DFM 2012) in conjunction with PACT 2012.
- Power Measurement and Concurrency Throttling for Energy Reduction in OpenMP Programs – Allan Porterfield, Stephen L. Olivier, Sridutt Bhalachandra, and Jan F. Prins. 9th Workshop on High-Performance, Power-Aware Computing May 2013 Boston MA USA.

Publications in development include:

- Publication in development: HPX/APEX/RCR integration paper. International Workshop on Runtime and Operating Systems for Supercomputers (ROSS 2013), submission deadline March 15. (UO)

Next Steps – Year 2

Year 2 activities of the XPRESS project will include:

Component	Year 2 Tasks	Lead Institution(s)
OpenX Software Architecture	• Complete the OpenX architecture	SNL, IU
ParalleX Execution Model	• Finalize and document the new generation ParalleX execution model with locality management and introspective scheduling prioritization policies	IU
HPX-4 Runtime System	• Complete the HPX-3 to LXK stacking and design, and build new handler and LCO operators for HPX-4	IU
LXK Operating System	• Develop functional scalable version of OS for operation, testing, and evaluation • Integrating support for the Process Management Interface (PMI) into Kitten • Developing an implementation of the Parcels communication layer on top of Portals 4	SNL, LSU

Component	Year 2 Tasks	Lead Institution(s)
XPI	• Finalize API specification and implement first reference implementation with interface to HPX-3	IU
RIOS	• Publish specification of interface between OS and runtime system	SNL, IU
Compilation/ Introspection	• Develop and integrate contention/energy models into HPX and APEX • Improve/increase data sources for models by integrating into LXK • Finish design and start implementing multi-node data collection and contention/energy models	UNC/RENCI
APEX Performance Measurement	• HPX-3 ITT abstraction to use APEX measurement • Measurement wrapper libraries for XPI • Design measurement in APEX with full ParalleX context • More (full) integration with RCRToolkit • Develop performance data API for third-person observation of other XPRESS layers o XPI <-> HPX <-> OS • Design an APEX event-driven performance model	UO
Legacy Migration	• Finish the integration of OpenMP/MPI runtime with HPX as needed for step-1 migration and evaluating with benchmarks and applications • Initial implementation of OpenMP/MPI runtime using XPI interface • Adding those HPX/XPI features to the OpenUH OpenMP runtime that could be leveraged with the upcoming OpenMP 4.0 features. The work are parts of step-2 migration • Release a prototype implementation of OpenACC compiler in OpenUH	UH
Experiments and evaluation	• Explore with application codes in XPI, MPI, and OpenMP on top of the OpenX software stack	
Applications	• Conduct initial ports of test applications including Co-design center proxy apps	
Documentation	• Publish specification reports for XPI, ParalleX, and RIOS, with Principles of Operation for LXK and HPX-4	