An Early Prototype of an Autonomic Performance Environment for Exascale

Kevin Huck, Sameer Shende, Allen Malony
University of Oregon
5294 University Of Oregon
Eugene, Oregon 97403
{khuck,sameer,malony}@cs.uoregon.edu

Hartmut Kaiser
Center for Computation & Technology
Louisiana State University
216 Johnston Hall
Baton Rouge, LA 70803
hartmut.kaiser@gmail.com

Alan Porterfield, Rob Fowler
RENCI
100 Europa Dr.
Chapel Hill, NC 27517
{akp,rjf}@renci.org

Ron Brightwell
Sandia National Lab
PO Box 5800
Albuquerque, NM 87185
rbbrigh@sandia.gov

ABSTRACT

Extreme-scale computing requires a new perspective on the role of performance observation in the exascale system software stack. Because of the anticipated high concurrency and dynamic operation in these systems, it is no longer reasonable to expect that a post-mortem performance measurement and analysis methodology will suffice. Rather, there is a strong need for performance observation that merges first- and third-person observation, in situ analysis, and introspection across stack layers that serves online dynamic feedback and adaptation. In this paper we describe the DOE-funded XPRESS project and the role of autonomic performance support in exascale systems. XPRESS will build an integrated exascale software stack (called *OpenX*) that supports the ParalleX execution model and is targeted towards future exascale platforms. An initial version of an autonomic performance environment called *APEX* has been developed for OpenX using the current TAU performance technology and results are presented that highlight the challenges of highly integrative observation and runtime analysis.

1. INTRODUCTION

The challenges to achieving extreme-scale computing for DOE mission-critical applications demand basic research in future system software and programming models if strong-scaled problems today will derive performance advantage from Moore's Law and a broader range of problems will be able to fully exploit Exascale computing capabilities by the end of this decade. We are in the first year of a three-year DOE-funded X-Stack project, *eXascale PRogramming Environment and System Software (XPRESS)*, to investigate programming methods, runtime software, system services, and tools needed to create a proof-of-concept system software stack for Exascale, focused on strong-scaled computing for real-world applications. The XPRESS software development will be enabled and driven by innovative concepts in future extreme-scale computing and research to explore and evaluate them. These two project thrusts – concepts research and software stack development – will be coordinated and mutually supportive. XPRESS will engage the talents and contributions of Indiana University, Louisiana State University, University of Houston, University of Delaware, Oak Ridge National Laboratories, University of Oregon, and University of North Carolina at Chapel Hill to provide comprehensive coverage of the interrelated domains, disciplines, and advances necessary to inform and facilitate the research in Exascale computing, applications, systems, and programming.

What makes XPRESS unique as a project is that it will deliver a new and complete system software architecture based on the innovative *ParalleX* execution model [17]. ParalleX has been devised to address challenges of starvation, overhead, latency, and contention by enabling a new computing dynamic through the application of message-driven computation in a global address space context with lightweight synchronization. XPRESS will design an integrated exascale software stack for ParalleX, *OpenX*, and implement a critical proof-of-concept prototype to validate its concepts for achieving high levels of parallelism and resource efficiency.

One of the key components of the XPRESS project is a new approach to performance observation, measurement, analysis and runtime decision making in order to optimize performance. The particular challenges of accurately measuring the performance characteristics of ParalleX applications requires a new approach to parallel performance observation. The standard model of multiple operating system processes and threads observing themselves in a first-person manner while writing out performance profiles or traces for offline analysis will not adequately capture the full execution context, nor provide opportunities for runtime adaptation within OpenX. The approach taken in the XPRESS project is a new performance measurement system, called *APEX (Autonomic Performance Environment for eXascale)*. APEX will include methods for information sharing between the layers of the software stack, from the hardware through operating and runtime systems, all the way to domain specific or legacy applications. The performance measurement components will incorporate relevant information across stack layers, with merging of third-person performance observation of node-level and global resources, remote processes, and both operating and runtime system threads.

We have taken the first steps at realizing APEX with the current OpenX runtime system, *HPX*, the TAU Performance System, and the RCRToolkit. Results on these initial efforts are reported on below. First, an overview of the XPRESS project is given, followed by a description of OpenX components. We then discuss our APEX prototype implementation. Our objective is to learn from experiments with the software about the issues involved and how the relative software components need to be evolved to address them. The prototype is destined to be discarded, but the lessons learned will directly inform the final APEX version.

2. XPRESS OVERVIEW

2.1 Vision

The XPRESS project envisions a class of systems, both software and hardware, that reflects the new realities of emerging enabling component technologies and responds to the challenges they impose in terms of starvation, latency, overheads and contention, as well as the availability constraints of reliability, power consumption, and application user programmability. Innovative concepts and their synthesis are imperative if sustained exaflops performance is to be realized and applied for DOE mission-critical requirements. Future extreme-scale computational systems will move from previous static largely compiler and user based resource management to future dynamic runtime based system and work supervision with an entirely new form of lightweight kernel operating system exhibiting a single system image but comprising an ensemble of potentially millions of operating system agents operating in synergy and through a revolutionary relationship with runtime control.

For the first time, the tension between machine responsibilities and applications requirements will be balanced by a system incorporating this new dynamic runtime versus operating system relationship. The logical point of exchange is the *Prime Medium*, a new system-wide interface protocol that supports bi-directional information exchange including local and global state and mutual requirements of each other. Instrumentation measurements, faults, power consumption, utilization and demand, changes in requirements, locality attributes, and parallelism discovery are all aspects of the new vision of computing that will be enabled by the symbiotic relationship to be established between the runtime and operating software systems. But the vision must and does extend further to the programming models and methods that must provide the end user with abstractions and means of eliciting 10,000 times more parallelism than conventional practices and the ability to exploit runtime information and instrumentation for resource usage. New APIs both low level and domain specific will be required and provided to work in context of the new class of Exascale systems even as means of legacy mitigation allow old applications to run on new hardware/software system platforms. Among the interstices of these concepts and the system modules that manifest them is key support and new functionality to be attributed to and supported by advanced compilation methods.

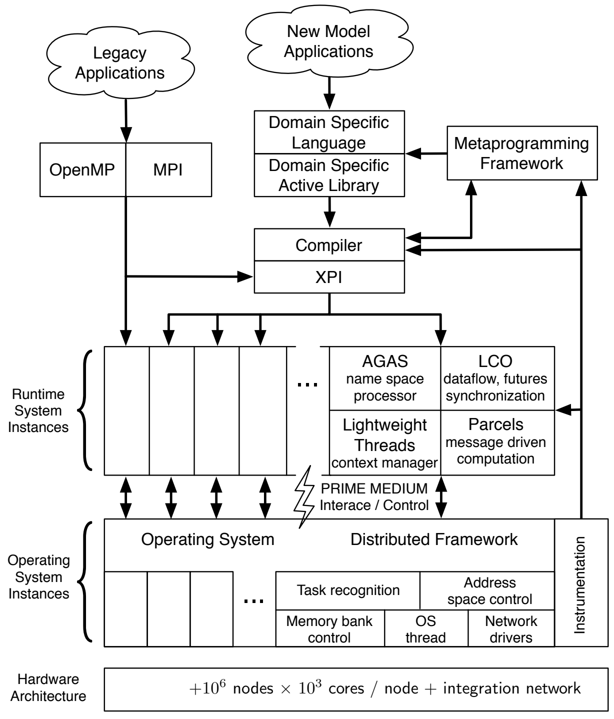

Legacy Applications
New Model Applications
OpenMP
MPI
Domain Specific Language
Domain Specific Active Library
Metaprogramming Framework
Compiler
XPI
Runtime System Instances
AGAS name space processor
LCO dataflow, futures synchronization
Lightweight Threads context manager
Parcels message driven computation
PRIME MEDIUM Interace / Control
Operating System Instances
Operating System
Distributed Framework
Task recognition
Address space control
Memory bank control
OS thread
Network drivers
Instrumentation
Hardware Architecture
$+10^6$ nodes $\times$ 10^3 cores / node + integration network

Figure 1: Major components of the OpenX architecture stack.

2.2 XPRESS Project

The XPRESS project is comprised of four major thrusts: system software, programming models and languages, applications, and crosscutting issues. Efforts in these areas will be defined and guided by key metrics.

2.2.1 *Exascale System Software*

The HPX-3 runtime system provides an early starting point as a programming tools and operating system target at the beginning of the XPRESS project. This will be phased out as HPX-4 - based on a new modular software architecture, OpenX - is developed incorporating added functionality for fault tolerance and power management to provide a robust community-grade runtime system. The LXK lightweight kernel operating system will be developed based on the advanced Kitten operating system [25, 5] in response to the new requirements for billion-way concurrency, introspective management of faults and power, assumed future directions of system architectures while dealing with near term systems, and management of protected and dynamic global virtual name space. LXK will be co-designed with HPX-4 around the centerpiece of the Prime Medium interface between the runtime and operating system software.

This interface will share information in both directions between the two major software layers for performance, reliability, and control of power consumption.

2.2.2 Programming Models and Languages

Two programming methods will be employed to provide early means of conducting application kernel driven experiments and to facilitate ease of programming and portability. A low-level imperative programming interface, XPI, will be developed to expose the semantic constructs comprising the ParalleX execution model embodied in the experimental HPX runtime system. To facilitate the creation of diverse domain specific programming libraries, the XPRESS project will develop a set of tools, a meta-programming framework, that will support rapid deployment of future embedded DSL formalisms exploiting the potential of ParalleX-based execution environments. The project will also develop an example of a DSL using the meta-DSL toolkit. The project will explore at depth and provide means of legacy mitigation to ensure seamless transition of legacy codes embodied with the MPI or OpenMP programming interfaces to the OpenX software stack. The project will develop and demonstrate interfaces to XPI for legacy code execution, provide for interoperability between software modules in both forms, and provide means for extending the parallelism incrementally within the MPI and OpenMP frameworks to increase scalability through the XPRESS OpenX software stack. APEX will provide instrumentation methods for performance measurement to XPI, DSL, and legacy codes.

2.2.3 Applications

Three computational science domains are included: climate science, nuclear energy, and plasma physics, represented by the Community Earth System Model (CESM) [30], Denovo [3], and GTC [19, 20, 7], respectively. GTC is a small code already being redesigned for ParalleX. CESM and Denovo are both large, complex applications that address important science and engineering questions with stringent numerical requirements, numerous configuration and science options, a need to retain the ability to run on modest computational resources, and a need to run on exascale systems. CESM and Denovo will be used to verify the ability to transition smoothly from the current MPI+OpenMP programming model to the new XPRESS runtime and programming models. A fourth driver is the collection of libraries in Trilinos [11], which service the needs of hundreds of applications within DOE and across the world.

2.2.4 Crosscutting Issues

Essential cross-cutting functionality includes automatic control and introspection, resilience, power management and heterogeneity. Power management software in combination with anticipated energy efficient hardware will achieve much greater resource utilization per joule while dramatically reducing data movement, a major source of power consumption, through active locality management. Resilience will be achieved through a combination of in-memory micro-checkpointing that temporarily stores selected past data in memory in case of failures until no longer required for future calculations, with a new compute-validate-commit cycle that defers side-effects until previous calculations have been determined correct thus isolating error propagation.

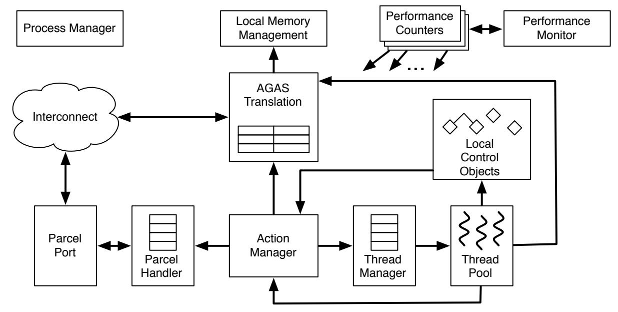


Figure 2: Modular structure of the HPX-3 implementation. HPX-3 implements the supporting functionality for all of the elements needed for the ParalleX model: AGAS (active global address space), parcel port and parcel handlers, HPX-threads and thread manager, ParalleX processes, LCOs (local control objects), performance counters enabling dynamic and intrinsic system and load estimates, and the means of integrating application specific components.

3. RELEVANT COMPONENTS

3.1 HPX

High Performance ParalleX (HPX [17, 16, 28, 2]) is the first open-source implementation of the ParalleX execution model. HPX is a modular, state-of-the-art runtime system developed for conventional architectures and, currently, Linux-based systems, such as large Non Uniform Memory Access (NUMA) machines and clusters. Strict adherence to Standard C++ 11 [29] and the utilization of the Boost C++ Libraries [4] makes HPX both portable and highly optimized. Its modular framework facilitates simple compile or runtime configuration and minimizes the runtime footprint. The current implementation of HPX supports all of the key ParalleX paradigms; Parcels and Parcel transport layer for inter-locality communication, PX-threads and their management, Local Control Objects (LCOs) for lightweight synchronization of medium grain parallelism, the Active Global Address Space (AGAS) as the basis to efficient resource management, and PX-processes to represent and manage the dynamic execution structure of an application at runtime.

The existing HPX-3 library (shown in Figure 2) supplies a valuable initial environment for the other parts of this project by providing immediate availability of stable programming interfaces and a functional, ParalleX-compliant low level runtime environment. Moreover, the experiences collected during several years of HPX development are highly relevant to the design, implementation, and optimization of all related XPRESS software layers, including the lightweight kernel, the distributed operating system, the improved next-generation runtime system HPX-4, and the development of migration path for legacy applications. The latter will take full advantage of the existing code by pairing it with interfaces that (a) permit direct access to core HPX functionality from traditional C and Fortran programs via lightweight programming interface (XPI) for advanced programmers who wish to explore it, and (b) enable traditional MPI and OpenMP functions by providing a thin translation layer for MPI combined with compiler support for OpenMP operation.

3.2 APEX

Scaling systems to extreme size capabilities is happening through three architectural trends. First, the number of computational cores on each chip and node will continue to increase. Second, the number of nodes will continue grow. Third, systems will become more heterogeneous at levels including accelerator modules (GPUs, MIC [14]), shared functional units on chip (IBM BlueGene and AMD Bulldozer), and performance heterogeneous multi-core chips based on a single instruction set architecture [18, 21]. The performance, energy, and health measurement and control infrastructure of XPRESS must address all of these.

Performance is no longer constrained primarily by the throughput of individual cores or the behavior of the code executing in a single pipeline. Increasingly cores interact with each other through the shared resources. These interactions are manifested by bottlenecks and queuing at scarce resources both on a chip (node), and between nodes. The system measurement and control components of XPRESS need to address all of these areas. Furthermore, to be useful for introspective adaptive control, both measurements and analyses need to be available in real time to software components at all levels.

Over the last ten years, high-performance computing (HPC) performance methods have evolved incrementally to serve the dominant architectures and programming models. The parallel performance abstractions embodied in HPC tools such as TAU [26], HPCToolkit [1], Open|SpeedShop [23], oprofile [15], and PAPI [6] are driven by the underlying system environment. Performance observation requirements have been satisfied by local measurements and offline performance optimization. The stable, static model has allowed performance tools to focus on a *first person* measurement model where performance is seen only by individual threads and collected at the end of execution for *post mortem* analysis. The first person approach is inadequate for exascale use since the shared resources operate independently of the cores with their own hardware monitors built in [13]. This requires using a system-wide *third person* measurement model. A key component of XPRESS will be a measurement and analysis facility designed specifically to support introspective adaptation for performance, energy, and reliability by making node-wide resource utilization data and analysis, energy consumption, and health information available in real time.

The entire exascale software stack needs to be performance-aware and performance-reactive, able to observe performance state holistically and to couple it with application knowledge for self-adaptive, runtime control. We use the term *autonomic performance* to reflect this idea. XPRESS will be designed and developed with an autonomic performance environment for exascale (called APEX) with this new performance paradigm.

The role of APEX in the XPRESS project will be to serve both top-down and bottom-up performance requirements of the OpenX software stack and its integrated operation. The top-down requirements are driven by the mapping of applications to the ParalleX model and the translation through the programming models and the language compilers into runtime operations and execution. The mapping and translation process at each level includes performance abstractions. Defining the set of parameters to be observed and then coupling the abstractions to the observables is the function of the hierarchical performance framework. The top-down view sees APEX functionality as part of application creation specifically to provide integrated performance assessment and tuning. APEX observability support will be built into each OpenX layer:

- LXK operating system will track system resource assignment, utilization, job contention, and overhead.
- HPX will track threads, queues, concurrency, remote operations, parcels, and memory management.
- ParalleX, DSLs and legacy codes will allow language-level performance semantics to be measured.

The APEX observation infrastructure will be specialized through programming to enable autonomic performance capabilities specific to the application. It is also important to highlight the importance of mapping high-level semantic context top-down to low-level observation where context can be used to distinguish between performance artifacts.

3.2.1 *Bottom-up Development Approach*

The bottom-up requirements of APEX are targeted to performance introspection across the OpenX layers to enable dynamic, adaptive operation and decision control. The working model here is multi-parameter system optimization. APEX creates the performance feedback mechanisms and builds an efficient infrastructure for connecting subscribers to runtime performance state. There are intra-level performance awareness needs for HPX and LXK, but these will clearly interplay with the overall application dynamics. The performance information provided for introspection at feedback points will be the result of runtime analysis. The design of APEX to meet both top-down and bottom-up requirements will enable closed-loop performance optimization for the ParalleX execution model. APEX development will be constrained by available exascale technology. There are important factors concerning the overhead of measurement, the cost of analysis, and the latency of feedback that will contribute to APEX's effectiveness. Our goal is to support the general concept of performance portability through dynamic adaptivity.

The bottom-up approach for XPRESS, will extend previous experimental work [8, 22, 24] on building decision support instrumentation (RCRToolkit) for introspective adaptive scheduling on current generation X86 64 Linux systems. In this approach, called Resource Centric Reflection, a system daemon (RCRdaemon) monitors shared, non-core resources, applies real-time analyses, and publishes both raw and processed information for use by other software components. The components include a display and logging tool (RCRtool) and an adaptive thread scheduler. Previously [8], performance was used with introspection to prevent performance degradation at very high loads in a commercial database system. In more recent work, the MAESTRO scheduler throttles thread/core concurrency to successfully reduce power consumption without degrading performance in several cases and in other cases has increased throughput. HPCToolkit [1] was modifided to use RCRToolkit information to distinguish cache misses that occur when there is a memory bandwidth bottleneck from those that occur when memory is lightly utilized. Since memory channels are monitored independently, RCRToolkit has been used to detect and diagnose memory imbalance problems.

Within a "node" software components will communicate through a self-describing blackboard structure. This is a shared memory region that is organized as a hierarchy of single-writer, multiple-reader data regions aligned along cache block and virtual memory page boundaries for performance and access control purposes. The single-writer, multiple-reader organization allows lock-free access and minimizes memory coherence overhead. The blackboard is self-describing, with a directory structure to locate regions owned by individual software modules as well as the data layout within each region. The blackboard is a key component of the *Prime Medium* communication between the operating and runtime systems.

Good individual node performance does not automatically result in good performance for hundreds of thousands of nodes. To understand and adapt application performance across the entire system will require tools that communicate with all of the single node blackboards to generate a global view of performance.

Conventional techniques used for off-line performance analysis such as profiling and tracing can generate a volume of data that will both impact application performance and require substantial computing power. To address this, we will build on past work with hierarchical lossy wavelet compression to filter out uninteresting parts of the performance signals while preserving outliers and anomalous patterns [10]. We have also used clustering [9] to further reduce data requirements for off-line analysis. These methods will extend the blackboard mechanism hierarchically to each level at which control decisions are made.

3.2.2 *Top-Down Development Approach*

In the APEX model performance observability requirements and semantics, starting with the application, are specified through the programming model and languages and implemented by instrumentation generated by the compiler that invokes an HPX performance measurement API. It couples the first person and third person performance views by translating application context down the OpenX stack in order to associate the execution performance state back up.

APEX will integrate TAU instrumentation capabilities with the XPRESS programmatic specialization methodology and its compiler framework, the legacy programming tools, and the imperative API for ParalleX-based system programming. TAU's mapping API will be used to create higher-level abstractions to contextualize lower-level measurements. Both static and dynamic wrapper interposition libraries will intercept and instrument the imperative API and the runtime layers of HPX. TAU's multi-level instrumentation API will be augmented with a performance feedback API that will supply vital runtime information to a compiler for further analysis.

4. APEX PROTOTYPE IMPLEMENTATION

In the first year of the ongoing XPRESS project, the project team has made significant strides 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.

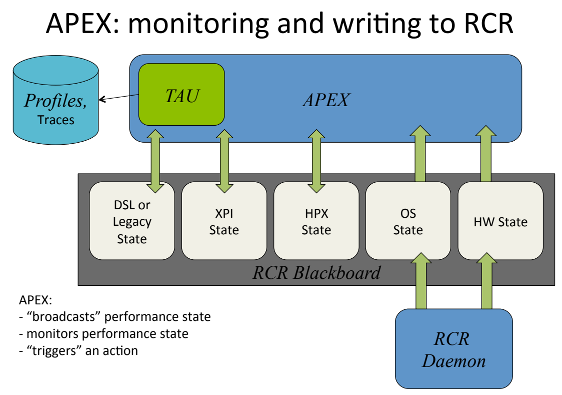


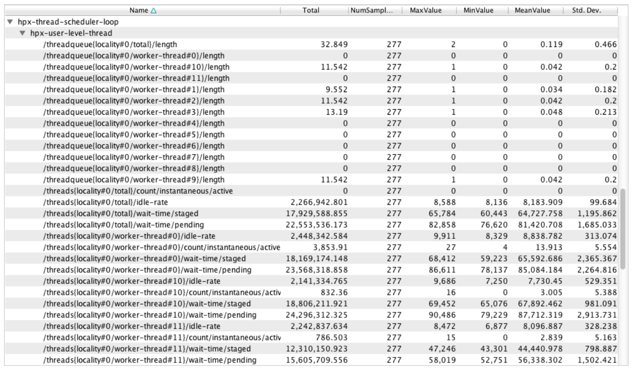
Figure 3: The APEX prototype architecture.

The XPRESS team has implemented an APEX prototype using TAU [27] 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. Figure 3 shows an architecture diagram of how APEX integrates with the layers of the OpenX stack.

The APEX prototype has been developed with the HPX-3 build process in mind, and is designed to integrate seamlessly into the build. If the APEX variables are not defined at the configuration step, 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.

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 only output to the user terminal or a log output, 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. Supporting the HPX-3 counters required a modification to TAU to create a new type of counter, a *context user event*. A context user event is annotated not only by the value it is measuring, but also the current execution context of the application, i.e. the current function or subroutine. 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 4 seconds during the run, resulting in 277 total samples. The measurement library also generates simple statistics (maximum, minimum, mean, variance). The context of the counters separates those collected during the thread scheduler part of the code from the application code.

The HPX-3 runtime thread manager was also instrumented



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 counters in TAU ParaProf.

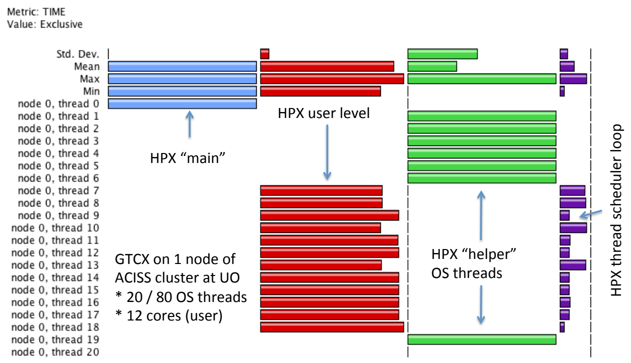
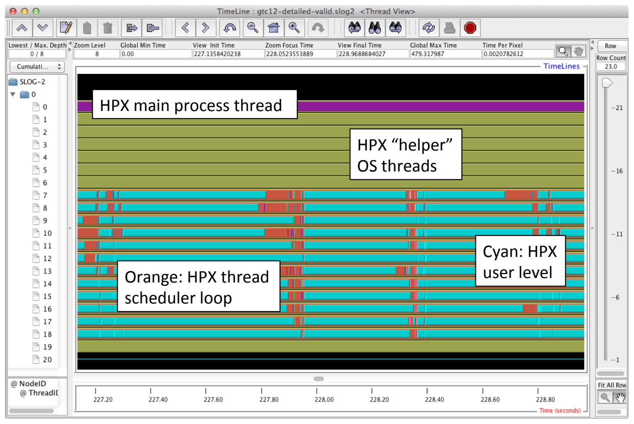


Figure 5: HPX-3 profile in TAU ParaProf.

with APEX timers. This instrumentation will 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 [31], the computational cluster at the University of Oregon. In addition to the twelve operating system threads that are directly involved in application execution progress, HPX-3 has eight additional operating system threads. Instrumenting HPX-3 in this way will help HPX developers in reducing the runtime thread scheduling overhead. Figure 6 shows a trace timeline of the thread scheduler in HPX-3 when executing an HPX-3 application.

Prior to the XPRESS project, HPX-3 had been instrumented with Intel Instrumentation and Tracing Technology (Intel® ITT) [12]. ITT provided detailed performance measurement of the HPX-3 runtime, not unlike the APEX implementation. However, ITT does not provide multi-node (multi-locality) support. In addition, there is no explicit annotation of whether an HPX thread completed its execution or was preempted and migrated to another thread (possibly even to another locality). Finally, there is no explicit annotation of the *provenance* of the HPX thread, i.e. the HPX thread which spawned it. By capturing this information, APEX can support more performance scenarios than ITT. To that end, The ITT instrumentation points have been extended to also provide detailed APEX instrumentation. Because APEX does provides multi-node support, multi-node performance experiments are possible with APEX.

One of the key aspects of the APEX third-person ob-



TimeLine : gtc12-detailed-valid.slog2 <Thread View>

Lowest / Max. Depth	Zoom Level	Global Min Time	View Init Time	Zoom Focus Time	View Final Time	Global Max Time	Time Per Pixel	Row Count
0 / 8	8	0.00	227.1358420238	228.0523553889	228.9688684027	479.317987	0.0020782612	23.0

Figure 6: HPX-3 trace in TAU Jumpshot.

servation 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. As desribed in § 3.2.1, 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 an 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.

5. EXPERIMENTAL RESULTS

GTC [19] is a 3D particle-in-cell (PIC) code developed for studying turbulent transport in magnetic confinement fusion plasmas [20, 7]. In the PIC method, the interaction between particles is calculated using a grid on which the charge of each particle is deposited and then used in the Poisson equation to evaluate the field. This is the scatter phase of the PIC algorithm. Next, the force on each particle is gathered from the grid-base field and evaluated at the particle location for use in the time advance. The PIC algorithm is the most expensive part of GTC.

Advances in multi-core technology have presented some challenges for GTC in terms of strong scaling. Strategies in combining OpenMP and MPI in GTC have not resulted in a clear solution to the scalability issue in the context of multi-core. This is a topic of active research. GTC has already been ported to C++ and HPX and algorithms are being modified to remove the global barriers which inhibit scalability.

To test the APEX prototype, HPX-3 was configured with APEX support, and the GTC sample application was built and executed on the ACISS cluster. The experiment was performed on a single node with 32 cores and 384GB of memory per node. Figure 7 shows the variation in the amount of

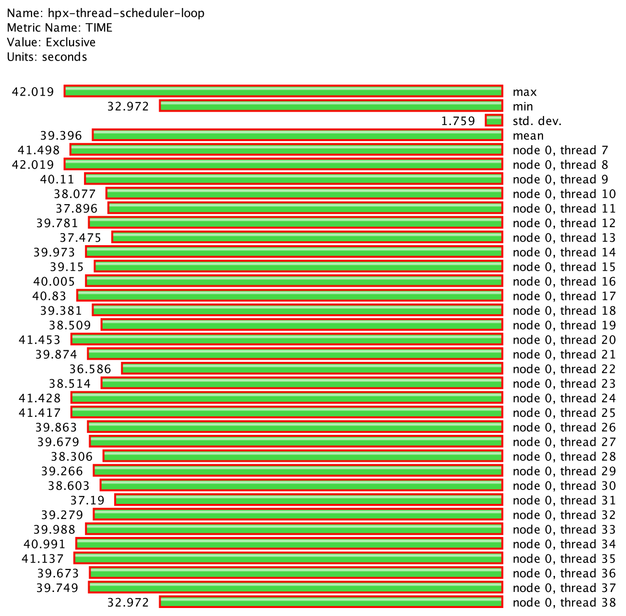


Figure 7: Performance profile of the HPX-3 thread scheduler loop while executing GTC. 32 HPX threads were requested, resulting in 40 operating system threads, total.

time spent in the thread scheduler loop. The overhead of the scheduler varies from 15% to 18.8% of the total execution time.

Because the overhead appears considerable for this application, more analysis is required. A trace of the same execution was also collected. Figure 8 shows a zoomed-in region of the trace timeline, showing the 32 HPX worker threads and 8 additional operating system threads. There appears to be synchronization artifacts in the application, causing the significant overhead in the scheduler. The synchronization occurs when work cannot be executed until data dependencies are resolved. In the trace figure, the cyan colored bars represent time spent in the `gtcx_partition_loop_action` task, while the orange bars in-between represent the time spent in `hpx-thread-scheduler-loop`. Further analysis is required, but it appears that this application performance can be improved if the "lockstep" behavior can be removed from the algorithm, or if its effect can be mitigated.

6. DISCUSSION AND FUTURE WORK

The initial APEX prototype will be useful in providing integrated performance measurement of HPX-3 applications running on current operating systems and platforms. However, there are a number of immediate issues and future work to discuss. The integration of HPX-3 counters into APEX, as discussed in § 4, will need to be modified so that they are not updated in APEX when periodically requested, but rather that they are updated directly when their values are changed. In addition, as discussed in § 4, APEX provides detailed instrumentation of the thread scheduler in HPX-3. However, like ITT, as yet there is no explicit annotation of whether an HPX thread completed its execution or was preempted and migrated to another thread, and there is no explicit annotation of the provenance of the HPX thread.

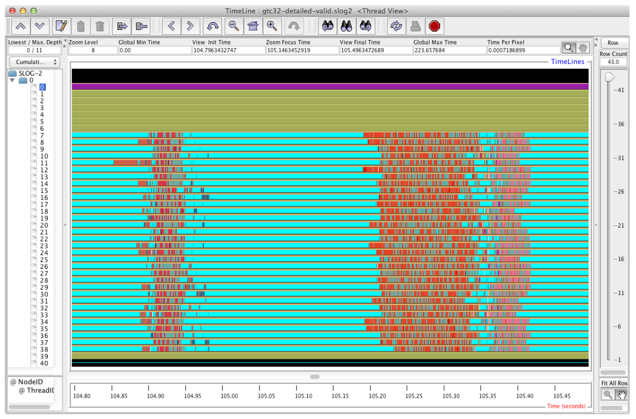


Figure 8: Performance trace detail of the HPX-3 operating system threads while executing GTC. The 32 runtime system threads are showing a synchronous pattern, limiting scalability and adding overhead.

Because TAU does not have the capability to capture that information explicitly, APEX will be designed to capture these important aspects of performance.

While the prototype implementation of APEX provides insight into HPX applications, there are some drawbacks to our 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 we still gain valuable insight into the application by implementing our prototype it in existing tools, running on existing systems, and mapping back to the ParalleX model. In addition, more components of the HPX-3 runtime can and should be instrumented, including the parcel transport layer, local control objects, and the active global address space.

Finally, there is much work to be done with respect to information sharing between components, runtime analysis and distillation of wide-scale performance data, integration with the operating system, measuring legacy and XPI level application context, and an event-based model for triggering runtime performance corrections. Over the lifetime of the XPRESS project, APEX will grow to provide these features.

7. ACKNOWLEDGEMENTS

This work is supported by grant number DE-SC0008638 from the United States Department of Energy Office of Science Advanced Scientific Computing Research X-stack Program. The ACISS cluster is supported by a Major Research Instrumentation grant from the National Science Foundation, Office of Cyber Infrastructure, grant number OCI-0960354.

8. REFERENCES

[1] Adhianto, L., Banerjee, S., Fagan, M., Krentel, M., Marin, G., Mellor-Crummey, J., and Tallent, N. HPCToolkit: Tools for Performance Analysis of Optimized Parallel Programs. *Concurrency*

and Computation: Practice and Experience 22, 6 (2010), 685–701. http://hpctoolkit.org/.

[2] Anderson, M., Brodowicz, M., Kaiser, H., and Sterling, T. L. An Application Driven Analysis of the ParalleX Execution Model. *CoRR abs/1109.5201* (2011). http://arxiv.org/abs/1109.5201.

[3] Baker, C., Davidson, G., Evans, T. M., Hamilton, S., Jarrell, J., and Joubert, W. High performance radiation transport simulations: preparing for titan. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis* (Los Alamitos, CA, USA, 2012), SC '12, IEEE Computer Society Press, pp. 47:1–47:10.

[4] Boost: a collection of free peer-reviewed portable C++ source libraries, 2011. http://www.boost.org/.

[5] Brightwell, R., and Pedretti, K. An intra-node implementation of OpenSHMEM using virtual address space mapping. In *Proceedings of the Fifth Partitioned Global Address Space Conference* (October 2011).

[6] Dongarra, J., London, K., Moore, S., Mucci, P., and Terpstra, D. Using PAPI for hardware performance monitoring on linux systems. In *International Conference on Linux Clusters: The HPC Revolution* (June 2001).

[7] Ethier, S., Tang, W. M., and Lin, Z. Gyrokinetic particle-in-cell simulations of plasma microturbulence on advanced computing platforms. *J. Phys: Conf. Ser.16* (2005).

[8] Fowler, R., Cox, A., Elnikety, S., and Zwaenepoel, W. Using Performance Reflection in Systems Software. In *Proceedings of USENIX Workshop on Hot Topics in Operating Systems (HOTOS IX)* (Lihue, HI, Mar. 2003). Extended abstract.

[9] Gamblin, T., de Supinski, B., Schulz, M., Fowler, R., and Reed, D. Efficiently clustering performance data at massive scales. In *Proceedings of the International Conference on Supercomputing 2010 (ICS2010)* (Tsukuba, Japan, June 2010), ACM.

[10] Gamblin, T., de Supinski, B. R., Schultz, M., Fowler, R., and Reed, D. A. Scalable load-balance measurement for SPMD codes. In *Proceedings of Supercomputing 2008* (Austin, TX, Nov. 2008), ACM/IEEE.

[11] Heroux, M., Bartlett, R., Hoekstra, V. H. R., Hu, J., Kolda, T., Lehoucq, R., Long, K., Pawlowski, R., Phipps, E., Salinger, A., Thornquist, H., Tuminaro, R., Willenbring, J., and Williams, A. An Overview of Trilinos. Tech. Rep. SAND2003-2927, Sandia National Laboratories, 2003.

[12] Intel. Intel® ITT API open source version. `http://software.intel.com/en-us/articles/intel-itt-api-open-source`, 2013.

[13] Intel Corporation. *Intel(R) Xeon(R) Processor 7500 Series Uncore Programming Guide*, March 2010.

[14] Intel Corporation. Intel MIC. `http://www.intel.com/content/www/us/en/high-performance-computing/high-performance-xeon-phi-coprocessor-brief.html`, 2013.

[15] John Levon *et al.* OProfile. `http://oprofile.sourceforge.net/`. 14 April 2006.

[16] Kaiser, H., Adelstein-Lelbach, B., et al. HPX SVN repository, 2011. Available under a BSD-style open source license. Contact gopx@cct.lsu.edu for repository access.

[17] Kaiser, H., Brodowicz, M., and Sterling, T. ParalleX: An advanced parallel execution model for scaling-impaired applications. In *Parallel Processing Workshops* (Los Alamitos, CA, USA, 2009), IEEE Computer Society, pp. 394–401.

[18] Kumar, R., Tullsen, D. M., Ranganathan, P., Jouppi, N. P., and Farkas, K. I. Single-ISA heterogeneous multi-core architectures for multithreaded workload performance. *Computer Architecture, International Symposium on 0* (2004), 64.

[19] Lin, Z., Ethier, S., and Lewandowski, J. GTC: 3D Gyrokinetic Toroidal Code, 2012.

[20] Lin, Z., Hahm, T. S., Lee, W. W., Tang, W. M., and White, R. B. Turbulent transport reduction by zonal flows: Massively parallel simulations. *Science 281*, 5384 (1998), 1835–1837.

[21] Nvidia Corporation. The benefits of quad core CPUs in mobile devices. `http://www.nvidia.com/content/PDF/tegra_white_papers/tegra-whitepaper-0911a.pdf`.

[22] Olivier, S., Porterfield, A., Wheeler, K., and Prins, J. Scheduling task parallelism on multi-socket multicore systems. In *International Workshop on Runtime and Operating Systems for Supercomputers* (Tuson, AZ, USA, June 2011).

[23] Open|SpeedShop. http://www.openspeedshop.org/wp/.

[24] Porterfield, A., Fowler, R., and Lim, M. Y. RCRTool design document; version 0.1. Tech. Rep. RENCI Technical Report TR-10-01, RENCI, 2010.

[25] Sandia National Laboratories. *The Kitten Lightweight Kernel*. `https://software.sandia.gov/trac/kitten`.

[26] Shende, S., and Malony, A. The TAU Parallel Performance System. *International Journal of High Performance Computing Applications 20*, 2, Summer (2006), 287–311. ACTS Collection Special Issue.

[27] Shende, S., and Malony, A. D. The TAU Parallel Performance System. *International Journal of High Performance Computing Applications 20*, 2 (Summer 2006), 287–331.

[28] STE||AR Group. Systems Technologies, Emerging Parallelism, and Algorithms Reseach, 2011. http://stellar.cct.lsu.edu.

[29] The C++ Standards Committee. ISO/IEC 14882:2011, Standard for Programming Language C++. Tech. rep., ISO/IEC, 2011. http://www.open-std.org/jtc1/sc22/wg21.

[30] University Corporation for Atmospheric Research. Community Earth System Model (CESM). `http://www.cesm.ucar.edu`, 2013.

[31] University of Oregon. ACISS. `http://aciss.uoregon.edu`, 2013.