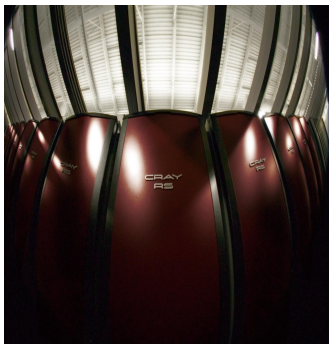
Exceptional service in the national interest

Sandia National Laboratories






eXascale Programming Environment and System Software (XPRESS)

Ron Brightwell, Technical Manager

Scalable System Software Department

U.S. DEPARTMENT OF ENERGY

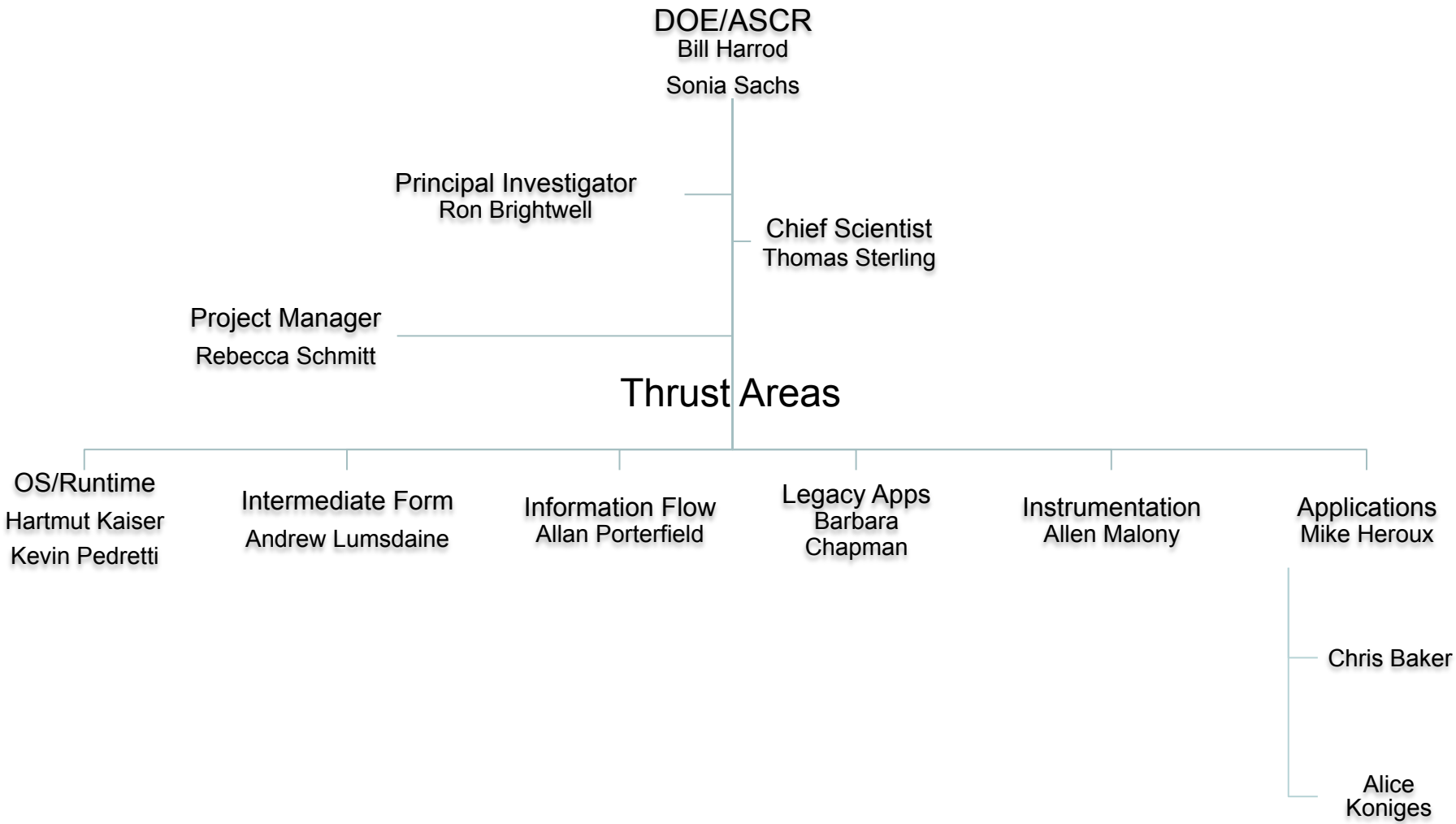
NNSA National Nuclear Security Administration

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

XPRESS Team

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

Organization Chart



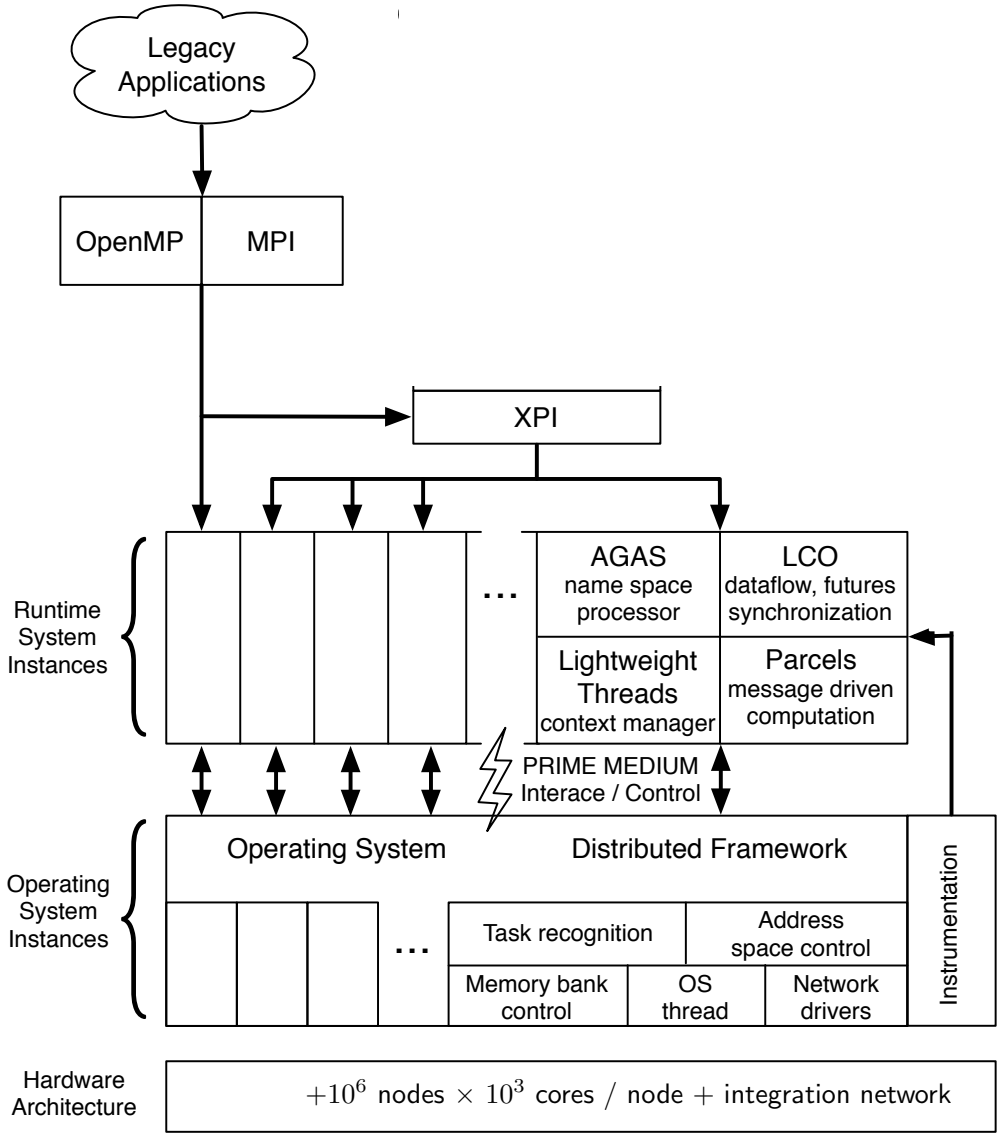
XPRESS Goals, Objectives, and Approach

- Goals:
  - Enable exascale performance capability for current and future DOE applications
  - Develop and deliver a practical computing system software X-stack , “OpenX”, for future practical DOE computing systems
  - Provide programming methods, environments, languages, and tools for effective means of expressing application and system software for portable exascale system execution
- Objectives:
  - Derive a dynamic adaptive introspective strategy for exploiting opportunities and addressing critical exascale technology challenges in the form of an abstract execution model
  - Devise a software architecture as a framework for future exascale system design and implementation
  - Implement core interrelated and interoperable components of the software architecture to realize a fully working and usable system
  - Test, evaluate, validate, and demonstrate correctness, performance, resiliency, and energy efficiency
  - Provide technology transfer through cooperative engagement of industry hardware and software vendors and national labs via documentation and training
- Approach
  - Research, develop, and deploy a software stack to exploit the ParalleX execution model

ParalleX Execution Model

- An execution model to provide the governing principles of computation to guide the system codesign and interoperability of software component layers and portability across system classes
- Goal is to provide conceptual foundation to dramatically increase efficiency and scalability through transition from static to dynamic resource management and task scheduling and exploitation of new sources of parallelism
- Key semantic constructs
  - Active Global Address Space ("AGAS") for single system image
  - First class lightweight user threads for medium-gain parallelism
  - "Parcels" message-driven computing for latency mitigation
  - Local Control Objects ("LCO") for powerful system synchronization
- Performance strategy
  - Scalability through lightweight thread level parallelism, overlapping successive phases of computation with powerful synchronization and elimination of global barriers, automatic exposure/ exploitation of intrinsic meta-data parallelism, effective use of finer grain parallelism through reduction of overhead that bounds granularity
  - Latency mitigation through parcels by reducing # of long distance actions (split-phase transactions), localizing remote data and work, migrating continuations to change locus of continued execution with data structure, lightweight thread context switching, and direct in-memory and parcel generation without thread instantiation, and locality semantics
  - Overhead reduction is derived by powerful semantics of synchronization for minimum work, optimized thread control
  - Contention amelioration through dynamic resource management

Software Architecture



LXK – Lightweight eXascale Kernel

- Based on Sandia's Kitten lightweight kernel
  - Boots identically to Linux
  - Repurposes basic Linux functionality (PCI, NUMA, ACPI, etc.)
  - Supports POSIX threads (NPTL) and OpenMP
  - Allows innovation in key areas
    - Memory management
    - Multicore messaging
    - Network stack optimizations
    - Fully tick-less operation
  - 20K LOC
- Allows for re-thinking OS structure and implementation of
  - Lightweight asynchronous system services
  - Dynamic composability and modularity
  - Adaptive resource policy enforcement mechanisms
  - Interface to runtime system(s)
  - Integrated instrumentation and monitoring

Kitten Implementation

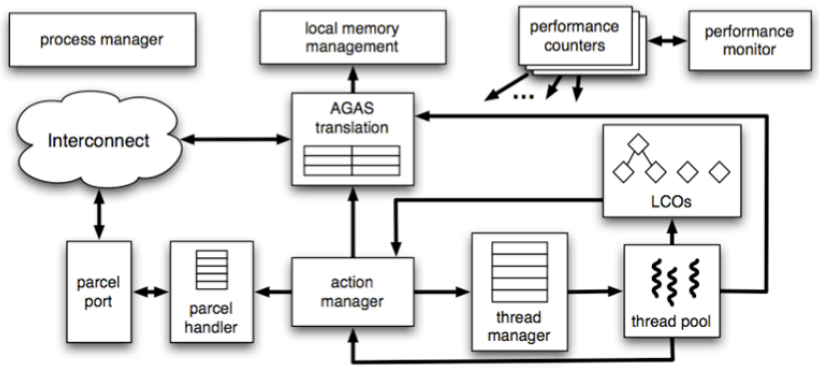
- Monolithic, C code, GNU toolchain, Kbuild configuration
- Supports x86-64 architecture only, considering port to ARM
  - Boots on standard PC architecture, Cray XT, and in virtual machines
  - Boots identically to Linux (Kitten bzImage and init_task)
- Repurposes basic functionality from Linux
  - Hardware bootstrap
  - Basic OS kernel primitives (lists, locks, wait queues, etc.)
  - PCI, NUMA, ACPI, IOMMU, …
  - Directory structure similar to Linux, arch dependent/independent dirs
- Custom address space management and task management
  - User-level API for managing physical memory, building virtual address spaces
  - User-level API for creating tasks, which run in virtual address spaces
  - User-level API for migrating tasks between cores

Kitten Thread Support

- Kitten user-applications link with standard GNU C library (Glibc) and other system libraries installed on the Linux build host
- Functionality added to Kitten to support Glibc NPTL POSIX threads implementation
  - Futex() system call (fast user-level locking)
  - Basic support for signals
  - Match Linux implementation of thread local storage
  - Support for multiple threads per CPU core, preemptively scheduled
- Kitten supports runtimes that work on top of POSIX threads
  - Glibc GOMP OpenMP implementation
  - Sandia Qthreads
  - Probably others with a little effort

HPX Runtime System

- A next-generation runtime system software layer that supports the semantics of the ParalleX execution model for significant increase in efficiency and scalability
- HPX-3 provides
  - Existing early proof-of-concept software dynamic adaptive resource management, task scheduling, global name space, efficient powerful synchronization, and Parcel message-driven computation. Interfaces with conventional Unix-like OS.
- HPX-4
  - Modular system software architecture with specified functionality, interfaces and protocols for intra-operability and interfaces to OS and programming environment
  - Introspection closed-loop system for
    - Resiliency through microcheckpointing
    - Dynamic load balancing
    - Power monitoring and control



XPI – Low-Level Intermediate Form

- User programming syntax and source-to-source compiler target for high-level programming languages
- Provides stable application platform based on HPX, which is expected to change underneath throughout project
- A library of C-bindings to represent lowest-level semantics, policies, and mechanisms for ParalleX execution model
- XPI construct classes
  - Process
  - Thread
  - Locality
  - Parcel
  - Future
  - Dataflow
  - Housekeeping

XPRESS Information Flow

- Passing information between layers will be critical
  - In current systems information flows one direction
- For Exascale static scheduling decisions will not work
  - Dynamic environment
    - Billion-way parallelism
    - Resilience
    - Reliability
    - Energy
    - Shared resource contention
- Feedback will be required

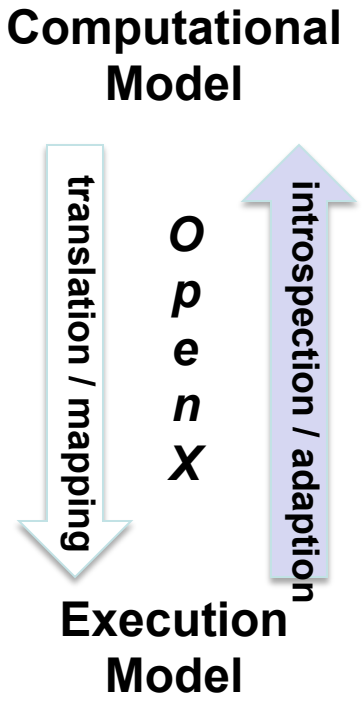
Performance Information as Glue

- Performance information
  - Current – post-execution performance tools
  - Exascale – dynamic application introspection

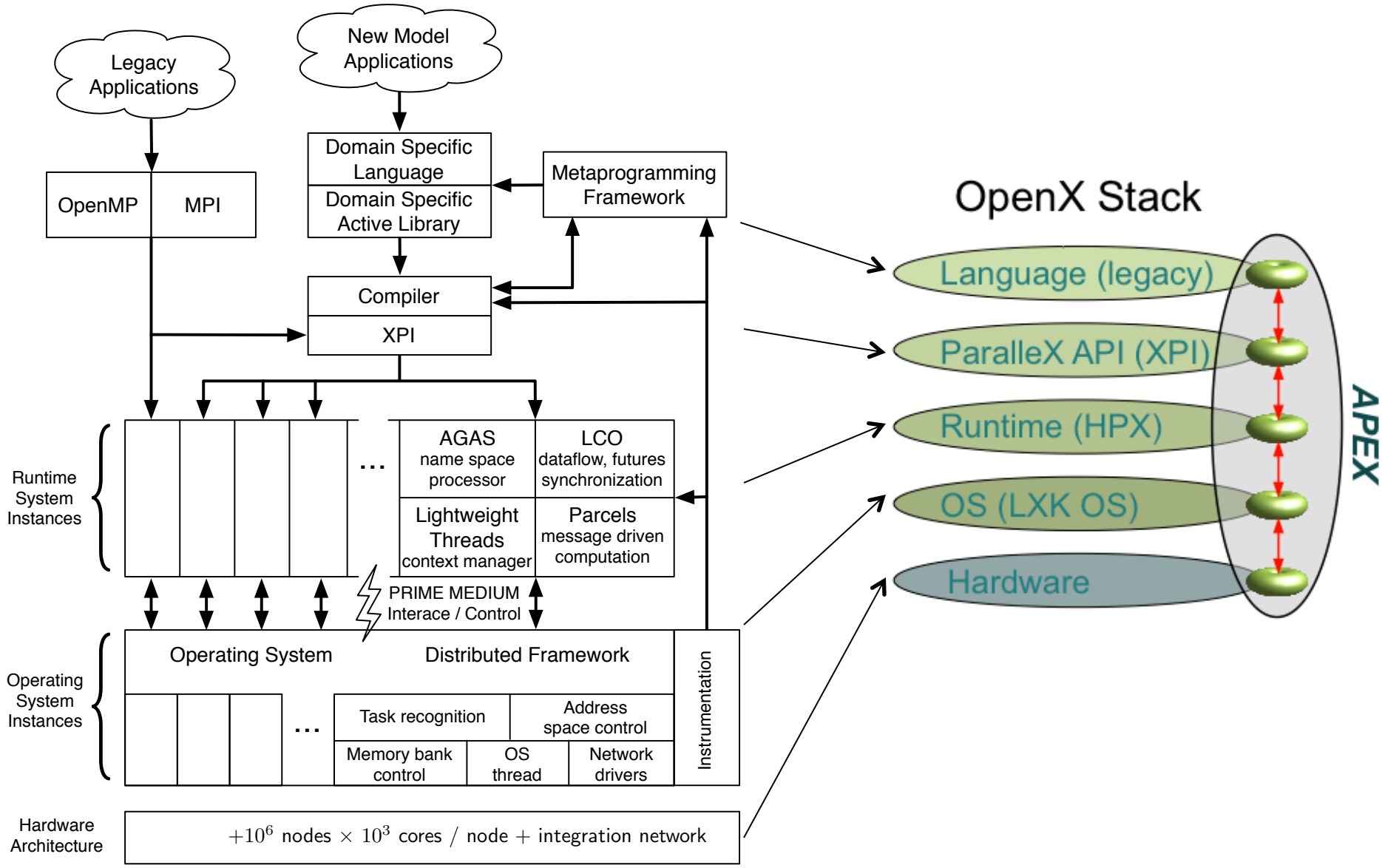
- For performance and reliability thread/core/node/system knowledge will be critical throughout the software stack
  - Interfaces designed to enable information flow
    - Utilities need to know current system performance
    - Utilities need to know how other utilities are reacting

Exascale Performance Observability

- Exascale requires a fundamentally new observability paradigm
  - Reflects translation of application and mapping of computation model to execution model
  - Designed specifically to support introspection of runtime performance for adaptation and control
  - Aware of multiple objectives
    - System-level resource utilization data and analysis, energy consumption, and health information
- Exascale observability abstraction
  - Inherent state of exascale execution is dynamic
  - Embodies non-stationarity of performance, energy, resilience during application execution
  - Constantly shaped by the adaptation of resources to meet computational needs and optimize execution objectives



OpenX Software Stack and APEX



APEX

- XPRESS performance measurement/analysis/introspection
  - Observation and runtime analysis of performance, energy, and reliability
  - Online introspection resource utilization, energy consumption, and health information
  - Coupling of introspection with OpenX software stack for self-adaptive control
- APEX: Autonomic Performance Environment for eXascale
  - Support performance awareness and performance reactivity
  - Couple with application and execution knowledge
  - Serve top-down AND bottom-up requirements of OpenX
  - Performance overlay on OpenX

APEX's Role for Top-Down OpenX Requirements

- Top-down requirements are driven by:
  - Mapping of applications to the ParalleX model
  - Translation through the programming models and the language compilers into runtime operations and execution
  - Performance abstractions (PA) at each level define:
    - set of parameters to be observed by the next levels down
    - Performance model to be evaluated and provide basis for control
  - Performance abstractions are coupled with observations through APEX's hierarchical performance framework
  - Realization of control mechanisms (reactive to PA actualization)
- Top-down view sees APEX functionality as part of the application's execution, specialized with observability and introspection support built into each OpenX layer:
  - LXK OS – system resource, utilization, job contention, overhead
  - HPX – threads, queues, concurrency, remote, parcels, memory
  - XPI/Legacy – language-level performance semantics

APEX's Role for Bottom-Up OpenX Requirements

- Bottom-up requirements are driven by:
  - Performance introspection across the OpenX layers
  - Enable dynamic, adaptive operation, and decision control
  - Online access and analysis of observations at different levels
  - Working model is multi-parameter system optimization
- APEX creates the performance feedback mechanisms and builds an efficient hierarchical infrastructure for connecting subscribers to runtime performance state
  - Intra-level performance awareness for HPX and LXK
  - Interplay with the overall application dynamics (top-down)
    - Top-down requirement implement constraints
  - Performance information is the result of runtime analysis

Top-Down Development Approach

- Define performance abstraction (focusing on higher level)
  - Specify observability requirements, and results semantics
  - Provide application context for association
  - What are the performance models, attributes, factors?
- Build into legacy languages and XPI
  - APEX programming of PA infrastructure
  - Invokes HPX performance measurement API
- Integrate TAU capabilities for APEX realization
  - Instrumentation
    - legacy programming (MPI, OpenMP, OpenACC)
    - imperative API (XPI) for ParalleX programming
  - TAU mapping for measurement contextualization
  - Wrapper interposition to intercept HPX runtime layer
- Create introspection API for feedback

Legacy Application Migration and Interoperability

- Seamless migration, no modification needed for apps
  - Retargeting OpenMP compiler to OpenX
  - Adapting MPI to OpenX, need MPI system-level adjustment, and change of configuration logic
- Two approaches for retargeting OpenMP via XPI:
  - Through a POSIX compliant interface using XPI, e.g. pthreads on top of XPI
    - +: No (or minor) modification needed to OpenMP compiler
    - -: May only use a limited subset of OpenX features
  - Rewrite OpenMP compiler and runtime to XPI
    - +: Explore OpenX advanced features
- OpenACC integration with OpenX
  - Adapt the OpenACC data movement mechanism to OpenX communication *parcels* in the *AGAS*
  - Integrate accelerator kernel execution of OpenACC with the OpenX execution model
- Evaluation:
  - Applications: Mini-apps with MPI+OpenMP/OpenACC
  - Performance and productivity feedback to the OpenX implementation team

XPRESS Team

Institution	Lead PI(s)	Major Focus
Sandia National Labs (Lead Institution)	Ron Brightwell (Lead PI) Mike Heroux (Application Lead)	• LXK operating system • OpenX software architecture • Applications
Indiana University	Andrew Lumsdaine (PI) Thomas Sterling (Chief Scientist)	• Parallex Execution Model • OpenX software architecture • HPX-4 runtime system • XPI
Louisiana State University	Hartmut Kaiser (PI)	• HPX-4 runtime system
University of Houston	Barbara Chapman (PI)	• Application migration
University of Oregon	Allen Maloney (PI)	• Performance instrumentation (APEX)
Oak Ridge National Lab	Chris Baker (PI)	• Applications
University of North Carolina at Chapel Hill/RENCI	Allan Porterfield (PI)	• XPI, HPX-4, APEX
Lawrence Berkeley National Lab	Alice Koniges (PI)	• Applications

Team Focus Areas

	IU	LSU	ORNL	SNL	UH	UNC	UO	LBL
HPX-4	✓	✓						
OpenX	✓	✓		✓			✓	
LXK				✓				
XPI	✓	✓				✓		
APEX						✓	✓	
Legacy Code Mitigation and Transition to OpenX		✓	✓	✓	✓			✓
Performance Measurements			✓	✓		✓	✓	✓