Exceptional service in the national interest

Sandia National Laboratories




ECI Runtime Systems Workshop Summary

Ron Brightwell

R&D Manager, Scalable System Software Department

U.S. DEPARTMENT OF ENERGY | NNSA

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

Purpose and Goals of the Workshop

- Review state-of-the-art in runtime systems (RTS)
- Identify challenges being addressed by current RTS R&D
- Identify research questions that need to be resolved
- Devise metrics, measures, benchmarks, and means for testing and evaluation for RTS prototypes
- Discuss R&D roadmap that will result in one or more high-quality RTS prototypes

- http://www.orau.gov/runtimesys2015/

Workshop Details

- March 11-13, 2015
- Rockville Hilton, Rockville, MD
- 45 domain experts in HPC runtime systems
- Content
  - Invited talks and breakout sessions
  - Topics
    - The architecture for future RTS software
    - RTS design
    - Outstanding research questions
    - Roadmap for the future

Focus Area 1: System Architecture

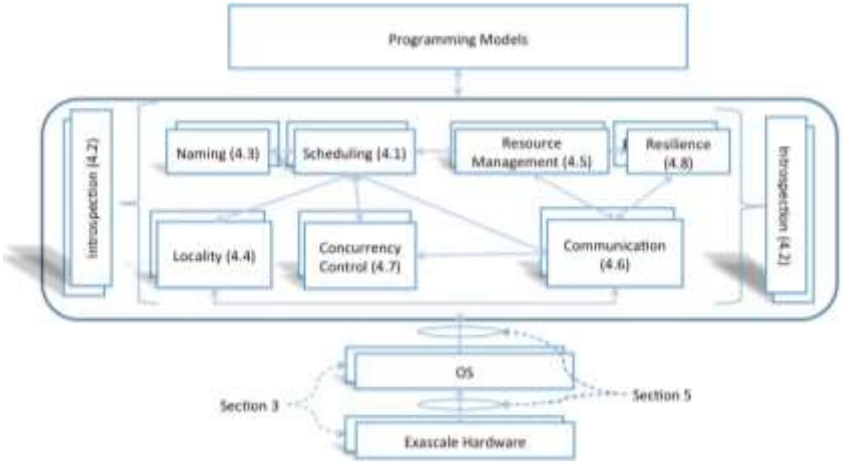
- Execution model
  - Governing principles of the strategy of computation
- Asynchrony
  - Semantics and control strategy in the presence of asynchrony
- System fragmentation
  - Scope of the RTS, from system- to node-level
- Relationship between operating system and RTS
  - Responsibilities and interfaces
- Relationship between programming models and RTS
  - Basic requirements for RTS
- Compile-time information, guidance, and constraints
  - Information that compilers can provide to the RTS
- Evaluation
  - Metrics for testing and evaluating a RTS

Focus Area 2: RTS Design

- Memory models, namespace, address space
  - How the RTS manages memory resources
- Introspection interfaces, policy, and control
  - How the RTS can use dynamic adaptive techniques
- Contribution to tools
  - Role of RTS in correctness analysis
- Parallelism forms, granularity, and synchronization
  - Role of RTS in managing parallelism
- Contribution and responsibility to reliability
  - Specific capabilities of the RTS for resilience
- Contribution and responsibility to power/energy
  - Role of RTS in minimizing energy costs
- Evaluation
  - How to test and evaluate RTS design

Workshop Builds on RTS Summit Activity

- RTS Summit meeting, April 9, 2014
- 11 attendees from X-Stack projects
- Day-long meeting to brainstorm about requirements for an exascale RTS
- Goal was to develop high-level requirements, roles, and responsibilities for RTS
- Provide some context for generating roadmap for future investments in RTS
- Services an RTS needs to provide
- Interfaces between RTS and
  - Node- and system-level hardware abstraction layers
  - Operating system
  - Programming interface
- Mapping these interfaces to existing RTS
- 45-page draft report

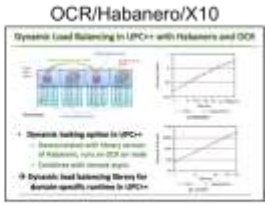


RTS R&D in Several ASCR Projects

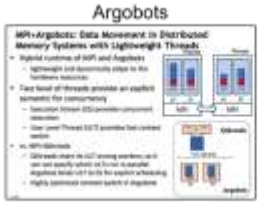
- X-Stack program has played key role in supporting RTS R&D for extreme-scale
- X-Stack renewal enables engagement across projects in RTS
  - Prototypes
  - Interfaces
  - Evaluation strategies

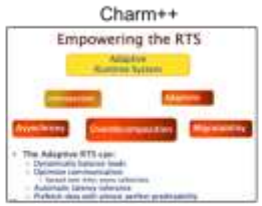
XPRESS

Traleika Glacier

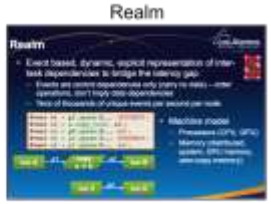


DEGAS





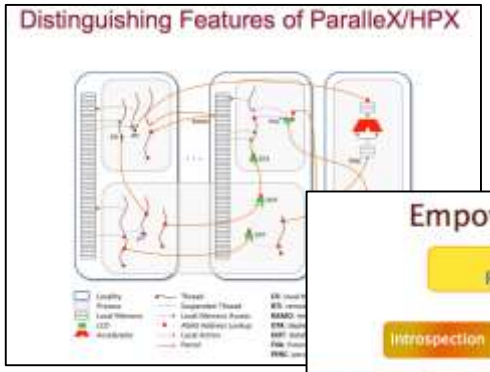
Argo



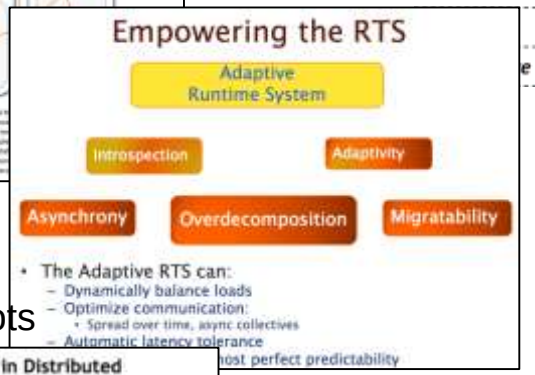
Legion

RTS in NNSA/ASC Projects

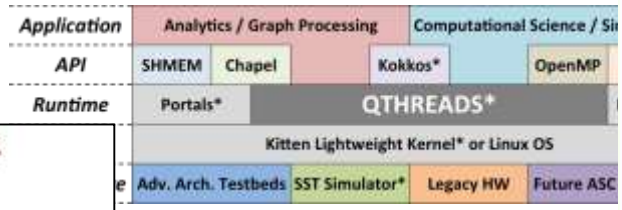
HPX



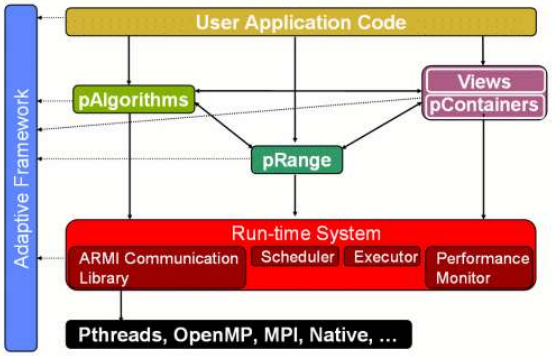
Charm++



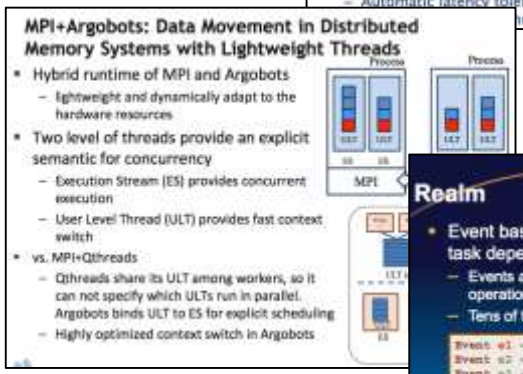
Qthreads



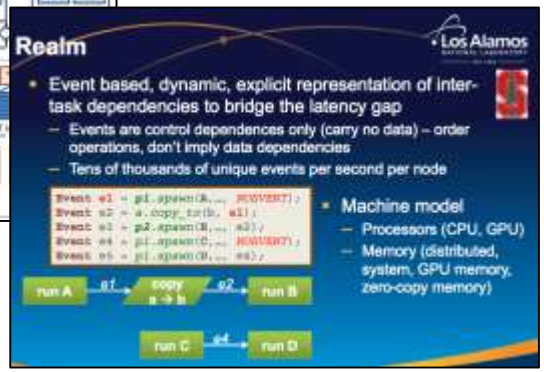
STAPL



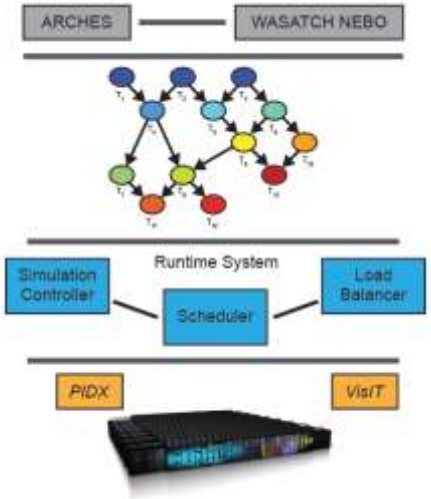
Argobots



Realm



UINTAH



Conductor

Many Other Run Time Systems

- Nanos/StarSS/OmpSS (BSC)
- StarPU/ForestGOMP (Inria)
- SWARM (ETI)
- MassiveThreads (U. Tokyo)
- Cilk/Cilk Plus (MIT/Intel)
- Grappa (UW/PNNL)
- HAS (AMD)

Definition of RTS (incomplete)

- Strong desire to understand responsibilities of the RTS
- Characteristics
  - Non-priveleged
    - Runs in application space
  - Ephemeral
    - Doesn't live beyond the application
  - Can manage hardware directly
    - As long as isolation and protection mechanisms are provided
  - Interfaces to the node-level OS
  - May interface to the system OS and the enclave OS

- Definition may be platform specific

Architecture for Exascale RTS

- Execution model
  - Struggle with nomenclature
  - Depends on what runtime service being provided
  - Runtime services should be able to be bypassed
- Asychrony
  - Performance variability – how to do resource management?
  - Some programming models embrace it
  - Everything needs to be lightweight – scheduling, synchronization, etc.
- Relationship between RTS and OS
  - Services used by application versus across applications
  - OS should still get out of the way but enable the RTS
- Relationship to PM
  - What gets exposed and what gets hidden (transparency)
  - Connection to services like data management, security, performance monitoring
  - Flow of information between app and RTS
- Evaluation
  - What are the metrics?
  - RTS portability

RTS Architecture (cont'd)

- Blurry lines between RTS above (PM) and below (OS)
  - Dynamic compilation, interpreted languages, etc. make this problem worse
- Lack of clear taxonomy is hindering effective integration
- Need requirements from the top
  - Loss of semantic information all the way down the stack
- QoS requirements, allocation of resources should be exposed as hints from the application programmer to drive policy decisions
- Managing shared resources
- Dealing with elasticity
- Resilience is a cross-cutting problem

RTS Design

- Memory System
  - Translation
  - Need to support static, semi-static, and dynamic use of memory
  - How to differentiate between memory and storage
  - How memory is virtualized
- Introspection
  - Need a well-defined set of policies and abstractions for reasoning about the behavior of the system
  - Need to be able to observe all aspects of the hardware
  - Different granularities of information to be observed
  - Cost of introspection
- Reliability
  - Vulnerability of the RTS to faults
  - Complexity of interactions exacerbates this problem
- Energy/Power Management
  - Responsibility of job scheduler, job-level RTS, node-level RTS

RTS Design (cont'd)

- Scheduling and Resource Management
  - Priorities
  - Load balancing
  - Latency hiding
  - Systems will be malleable and elastic
  - Resolving conflicts between different policies
- Tool Infrastructure
  - Toolchain needs to be co-designed with RTS
  - Attribution of performance bottlenecks
  - Interoperability of different programming systems and RTSs
  - Application developers need to understand detailed decisions by RTS
- Evaluation
  - Adoption is a good metric
  - Scalability, flexibility, portability, completeness, ease of use

Articulate the RTS Ecosystem

- Develop an ecosystem model for RTS components
- Determine which RTS services are stand-alone and which are embedded into larger components
  - RTS support for language-specific features
- Identify interfaces that are ready for a standardization process
- Process for transitioning RTS software from research to production

Metrics

- Don’t want performance metrics alone
  - Need relative metrics to evaluate research progress
  - Time to solution
  - Time to solution with failures
  - Time to solution with system variability
  - Time to solution under power/energy constraints
- Runtime overhead
  - CPU overhead
  - Memory overhead
- Portability of RTS
- Many concerns about
  - Evaluating the RTS (or PM)
  - Evaluating the implementation of the RTS (or PM)
  - Evaluating the ability of the hardware to support the RTS (or PM)

Dynamic Control

- What does each RTS layer or component control?
- How do layers coordinate toward goal-oriented optimizations?
- Need to identify resources that are managed
- Need to figure out how to coordinate and optimize across layers
- Backplane for communication between layers
- Define data and mechanisms for introspection

Resilience

- RTS needs to support resilience
  - Must interface to other software layers
- RTS also needs to be resilient
- RTS-based strategies
  - Task replication and migration
  - Fine-grain checkpointing
- Critical challenge for extreme-scale

Adoption

- New RTS layers must be done with application developers and system software developers
- DOE needs to partner with application teams
- Need to disseminate RTS R&D impact
  - Track open research questions
  - Share peer-reviewed success with broader community
- Co-design should include system software, applications, and platforms

Research Questions

- What are the forms of schedulable tasks managed by the RTS? (threads, processes, codelets, fibers, etc.)
- What is the assumed memory structure? What are the performance trade-offs and opportunities of dynamic allocation and redistribution?
- What are first-class objects that can be named and what is the scope of that name (locality)?
- Interfaces and flow of information involving RTS
- Control model for RTS introspection
- Managing overhead of hiding latency while exploiting parallelism
- What is the role of the RTS in reliability?
- Role of the RTS in managing power/energy
- Role of RTS in application interoperability
- What architectural support does the RTS need?
- How can performance modeling and evaluation be leveraged?

Research Questions (cont'd)

- RTS
  - User-level constructs that exist within a single executable
  - Part of the programming model implementation
  - Can the RTS support multiple PM/Es?
  - Can different RTSs use shared resources?
  - How does data move between runtimes?
- Convergence
  - No standard practice
  - Need to establish a process for incorporating research results into an initial production approach
  - View the RTS as a set of services and establish minimal set of services
  - Need an initial detailed survey and inventory of service/interface points
  - Allow for convergence on a few RTSs and establish attributes for interoperability

Research Questions (concl'd)

- Industry integration
  - How to incorporate research efforts to industry
- RTS characteristics
  - Are dynamic RTSs needed for exascale performance?
  - How much parallelism should be exposed to the RTS?
  - How should application communicate information about locality and load balance to the RTS?
  - How should the RTS interact with other parts of the system?

Key Takeaways

- Need to define a process to work through several issues
  - Workshop only scratched the surface
  - Need crisp definitions for basic terms
  - Need to agree on set of services to organize discussions
- Tension between monolithic approach and interoperable components
- Everyone wants control of the layers below them (including apps)
- Need bi-directional flow of information between layers
- Better agreement on what is “OS” and “RTS”
- Interoperability between different RTS
- Are dynamic RTS capable or necessary for exascale?
- Emerging awareness of ties between RTS and SSIO
- RTS itself will need to be resilient
- Introspection is a key aspect, but what can/should be queried?
- What is the path to production use? How to engage vendors?
- Need metrics, even to help with concepts (e.g., overdecomposition, dynamic)
- Will overheads outweigh benefits at scale?
- Need to catalog research questions that are being answered

Draft Report

- Currently 29 pages

2015 ECI Runtime Systems Workshop: Summary Report

March 11-13, 2015
Rockville Hilton, Rockville, MD

Background

Proposed exascale computing architectures present scientists with a number of challenges to reaching DOE scientific goals. Future runtime system software must achieve significant improvements in efficiency and scalability in the context of user productivity, performance portability, and dynamic adaptation.

In particular:

- Computing platforms must become significantly more responsive to power constraints, faults, and new goal-based programming models. Current system software is often very static in nature – computing jobs are given fixed numbers of compute resources at the beginning of every job, power is not dynamically adjusted to meet computational goals, and parallelism is often fixed.
- Runtime systems must support new, highly dynamic task-based programming environments that span computing resources inside a node as well as globally across the platform.
- New software frameworks supporting introspection, autonomic tuning, programing tools to support debugging and performance adaptation need runtime layers that can efficiently manage hierarchical memory, heterogeneous computing elements, and shared storage systems.

Advanced runtime systems must also be portable, have stable interfaces that can support the long development cycles of many computational science teams, and perform well across a variety of machines from different vendors or generations.

To address the research challenges outlined above, this workshop convened approximately 45 domain experts in High Performance Computing Runtime Systems (RTS) together for 2.5 days with the following high level objectives:

1. Propose, discuss, and determine the required characteristics of future extreme scale runtime systems
2. Devise metrics, measurements, benchmarks, and other means for testing and evaluation for prototypes of runtime systems,
3. Identify research questions that need to be resolved within the context of current experience and knowledge,
4. Discuss a research and development roadmap that will result in one or more high quality runtime system software packages that could be deployed in the 2023 timeframe, on extreme scale systems.