Why Nobody Should Care About Operating Systems for Exascale

Ron Brightwell
Scalable System Software
Sandia National Laboratories
Albuquerque, NM, USA

International Workshop on Runtime and Operating Systems for Supercomputers

May 31, 2011

Sandia is a Multiprogram Laboratory Operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy Under Contract DE-ACO4-94AL85000.

Outline

- Background
- DOE Exascale Initiative
- Exascale runtime systems
- Co-Design

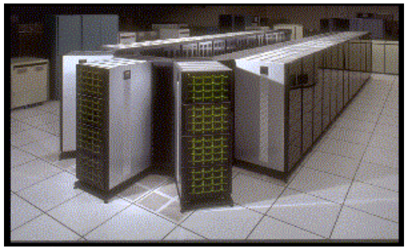
Sandia Massively Parallel Systems

1990

nCUBE2

- Sandia's first large MPP
- Achieved Gflops performance on applications

1993



Paragon

- Tens of users
- First periods processing MPP
- World record performance
- Routine 3D simulations
- SUNMOS lightweight kernel

1997

ASCI Red

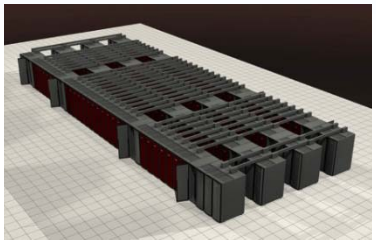
- Production MPP
- Hundreds of users
- Red & Black partitions
- Improved interconnect
- High-fidelity coupled 3-D physics
- Puma/Cougar lightweight kernel

1999

Cplant

- Commodity-based supercomputer
- Hundreds of users
- Enhanced simulation capacity
- Linux-based OS licensed for commercialization
- ~2000 nodes

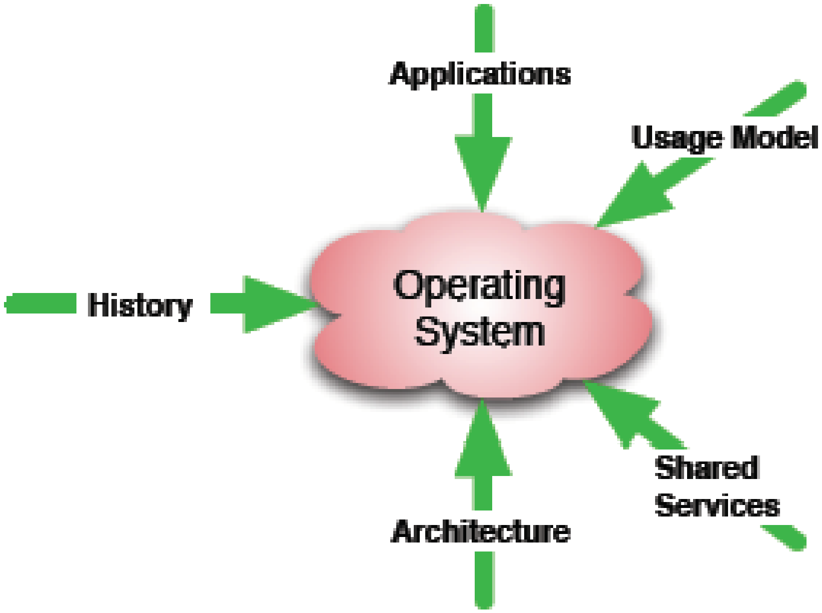
2004



Red Storm

- Prototype Cray XT
- Custom interconnect
- Purpose built RAS
- Highly balanced and scalable
- Catamount lightweight kernel
- Currently 38,400 cores (quad & dual)

Factors Influencing OS Design



- Lightweight OS
  - Small collection of apps
    - Single programming model
  - Single architecture
  - Single usage model
  - Small set of shared services
  - No history
- Puma/Cougar/Catamount
  - MPI
  - Distributed memory
  - Space-shared
  - Parallel file system
  - Batch scheduler

Sandia Lightweight Kernel Targets

- Massively parallel, extreme-scale, distributed-memory machine with a tightly-coupled network
- High-performance scientific and engineering modeling and simulation applications
- Enable fast message passing and execution
- Small memory footprint
- Persistent (fault tolerant)
- Offer a suitable development environment for parallel applications and libraries
- Emphasize efficiency over functionality
- Maximize the amount of resources (e.g. CPU, memory, and network bandwidth) allocated to the application
- Seek to minimize time to completion for the application
- Provide deterministic performance

Lightweight Kernel Approach

- Separate policy decision from policy enforcement
- Move resource management as close to application as possible
- Protect applications from each other
- Let user processes manage resources (via libraries)
- Get out of the way

Reasons for A Specialized Approach

- Maximize available compute node resources
  - Maximize CPU cycles delivered to application
    - Minimize time taken away from application process
    - No daemons
    - No paging
    - Deterministic performance
  - Maximize memory given to application
    - Minimize the amount of memory used for message passing
    - Kernel size is static
    - Somewhat less important but still can be significant on large-scale systems
  - Maximize memory bandwidth
    - Uses large page sizes to avoid TLB flushing
  - Maximize network resources
    - Physically contiguous memory model
    - Simple address translation and validation
      - No NIC address mappings to manage
- Increase reliability
  - Relatively small amount of source code
  - Reduced complexity
  - Support for small number of devices

Basic Principles

- Logical partitioning of nodes
- Compute nodes should be independent
  - Communicate only when absolutely necessary
- Limit resource use as much as possible
  - Expose low-level details to the application-level
  - Move complexity to application-level libraries
- KISS
  - Massively parallel computing is inherently complex
  - Reduce and eliminate complexity wherever possible

Quintessential Kernel (QK)

- Policy enforcer
- Initializes hardware
- Handles interrupts and exceptions
- Maintains hardware virtual addressing
- No virtual memory support
- Static size
- Non-blocking
- Small number of well-defined entry points

Process Control Thread (PCT)

- Runs in user space
- More privileged than user applications
- Policy maker
  - Process loading
  - Process scheduling
  - Virtual address space management
  - Fault handling
  - Signals
- Customizable
  - Singletasking or multitasking
  - Round robin or priority scheduling
  - High performance, debugging, or profiling version
- Changes behavior of OS without changing the kernel

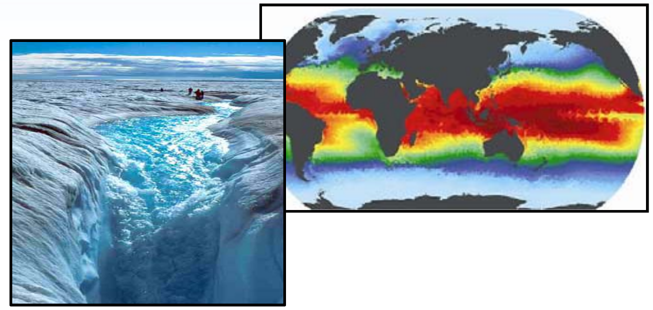
LWK Key Ideas

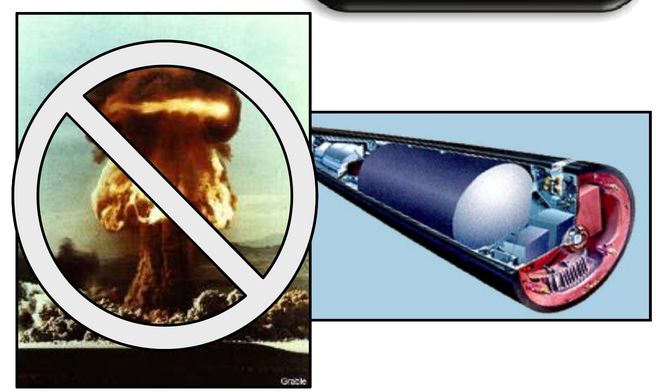
- Protection
  - Levels of trust
- Kernel is small
  - Very reliable
- Kernel is static
  - No structures depend on how many processes are running
- Resource management pushed out to application processes, libraries, and runtime system
- Services pushed out of kernel to PCT and runtime system

DOE Exascale Initiative

DOE mission imperatives require simulation and analysis for policy and decision making

- *Climate Change*: Understanding, mitigating and adapting to the effects of global warming
  - Sea level rise
  - Severe weather
  - Regional climate change
  - Geologic carbon sequestration
- *Energy*: Reducing U.S. reliance on foreign energy sources and reducing the carbon footprint of energy production
  - Reducing time and cost of reactor design and deployment
  - Improving the efficiency of combustion energy systems
- *National Nuclear Security*: Maintaining a safe, secure and reliable nuclear stockpile
  - Stockpile certification
  - Predictive scientific challenges
  - Real-time evaluation of urban nuclear detonation





Accomplishing these missions requires exascale resources.

Potential System Architecture Targets

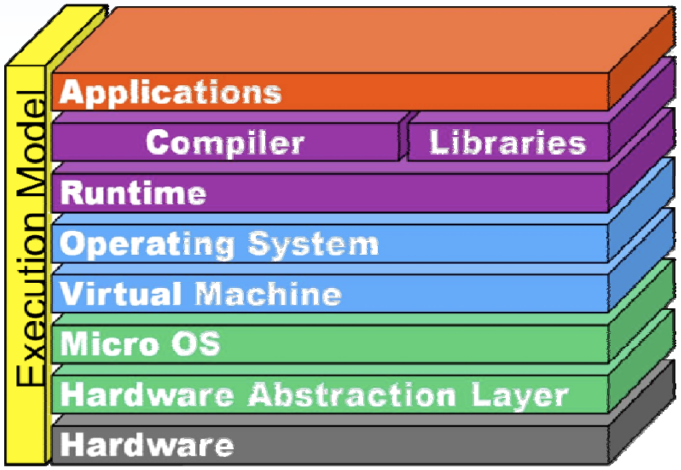
System attributes	2010	"2015-2018"		"2018-2020"	
System peak	2 Peta	200 Petaflop/sec		1 Exaflop/sec	
Power	6 MW	15 MW		20 MW	
System memory	0.3 PB	5 PB		32-64 PB	
Node performance	125 GF	0.5 TF	7 TF	1 TF	10 TF
Node memory BW	25 GB/s	0.1 TB/sec	1 TB/sec	0.4 TB/sec	4 TB/sec
Node concurrency	12	O(100)	O(1,000)	O(1,000)	O(10,000)
System size (nodes)	18,700	50,000	5,000	1,000,000	100,000
Total Node Interconnect BW	1.5 GB/s	20 GB/sec		200 GB/sec	
MTTI	days	O(1day)		O(1 day)	

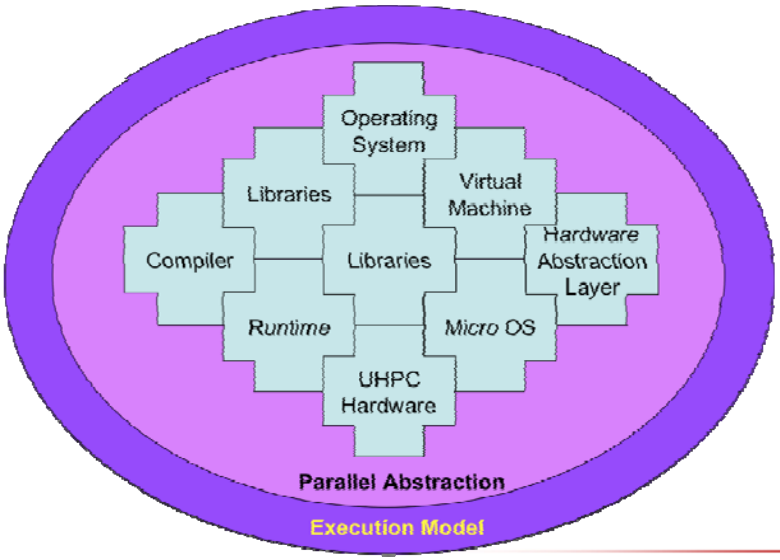
Investment in Critical Technologies is Needed for Exascale

- **System power** is a first class constraint on exascale system performance and effectiveness.
- **Memory** is an important component of meeting exascale power and applications goals.
- Early investment in several efforts to decide in 2013 on exascale **programming model**, allowing exemplar applications effective access to 2015 system for both mission and science.
- Investment in exascale **processor design** to achieve an exascale-like system in 2015.
- **Operating System** strategy for exascale is critical for node performance at scale and for efficient support of new programming models and run time systems.
- **Reliability and resiliency** are critical at this scale and require applications neutral movement of the file system (for check pointing, in particular) closer to the running apps.
- **HPC co-design strategy** and implementation requires a set of a hierarchical performance models and simulators as well as commitment from apps, software and architecture communities.

System software as currently implemented is not suitable for exascale system

- Barriers
  - System management SW not parallel
  - Current OS stack designed to manage only O(10) cores on node
  - Unprepared for industry shift to NVRAM
  - OS management of I/O has hit a wall
  - Not prepared for massive concurrency
- Technical Focus Areas
  - Design HPC OS to partition and manage node resources to support massively concurrency
  - I/O system to support on-chip NVRAM
  - Co-design messaging system with new hardware to achieve required message rates
- Technical gaps
  - 10X: in affordable I/O rates
  - 10X: in on-node message injection rates
  - 100X: in concurrency of on-chip messaging hardware/software
  - 10X: in OS resource management



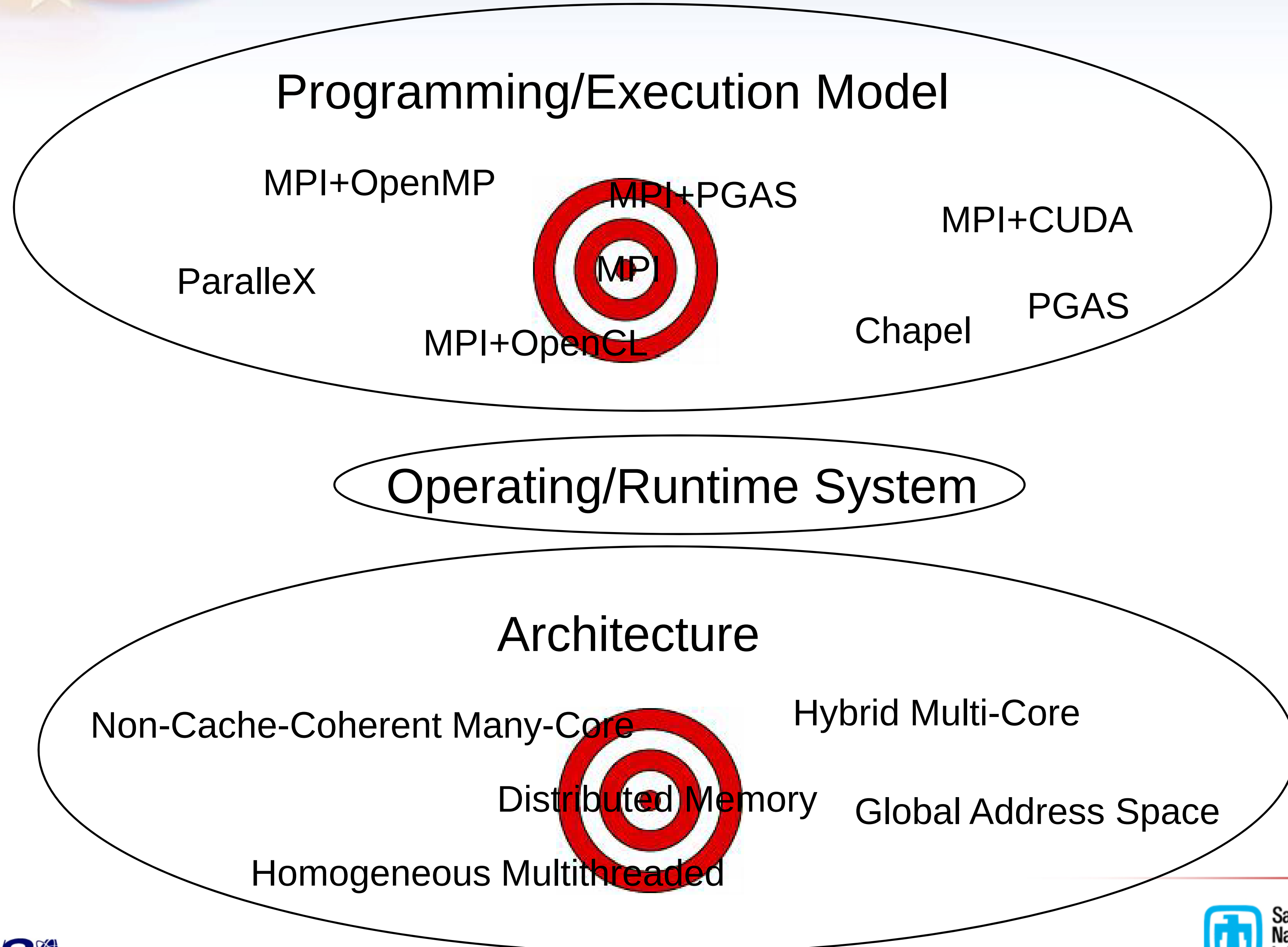


Software challenges in extreme scale systems, *Sarkar, 2010*

NNSA National Nuclear Security Administration

Sandia National Laboratories

Exascale Challenge for System Software



NNSA National Nuclear Security Administration

Sandia National Laboratories

Exascale Runtime Systems

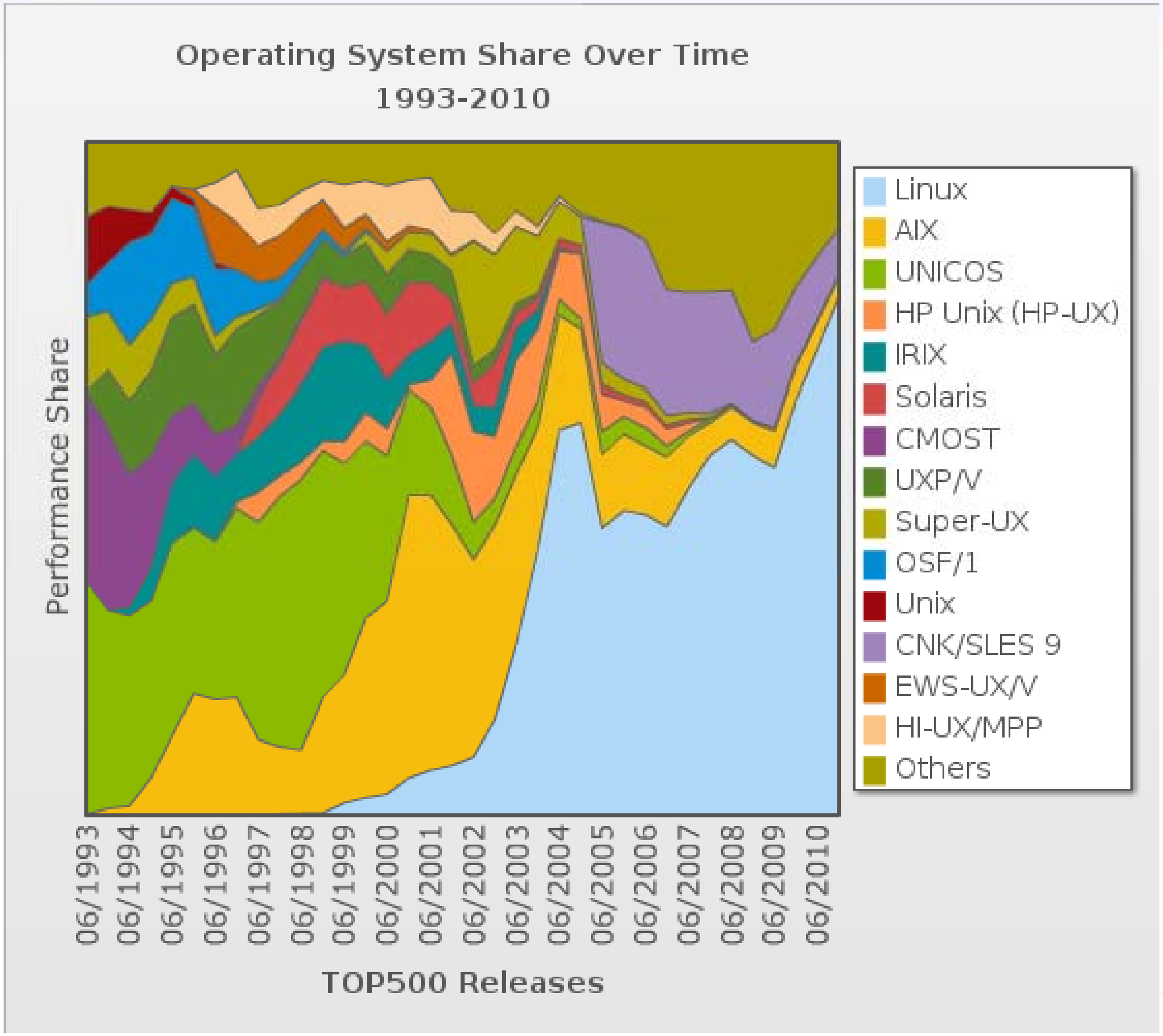
Pros and Cons of LWK Approach (From a Runtime Perspective)

- Cons
  - Node-level resource allocation and management is static
    - Memory allocation happens at application load time
    - Bad for shared memory on NUMA systems
  - Runtime components only communicate on set-up and tear-down
- Pros
  - Supports an application-specific runtime
    - Never happened in practice
    - OSFA worked for MPI applications
  - User-level networking
    - Runtime system can use same network interface as applications
    - No need for communication stack inside the OS
  - Memory management and scheduling are greatly simplified
    - User processes are allocated out of PCT heap

Forces Driving Exascale System Software

- Energy constraints and power management
  - Reduced data movement
- Resiliency
  - More frequent failures
- Concurrency
  - O(1k – 10k) threads per node
- Heterogeneity
  - Different types of cores
  - Non-coherent shared memory
  - Deeper memory hierarchies
- Highly unbalanced systems
  - Compute performance will dominate
- More complex applications
  - Dynamic, data-dependent algorithms
- Support for legacy interfaces and tools

Linux is the Dominant OS on the Top 500



Are These Really Linux Supercomputers?

- #1 - Tianhe-1A
  - 14,336 6-core Intel Xeons
    - 86,016
    - 3%
  - 7168 448-core Nvidia GPUs
    - 3,211,264 total cores
    - 97%

- #7 - Roadrunner
  - 6120 2-core AMD Opterons
    - 13,824 cores
    - 11%
  - 12,240 9-core IBM PowerXCell 8is
    - 116,640 cores
    - 89%

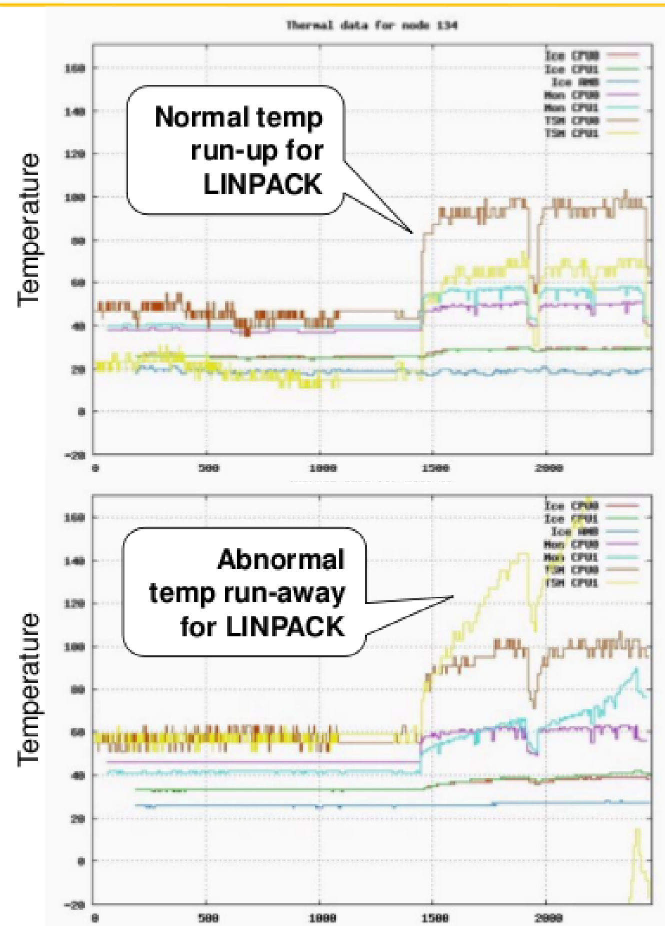
- Maybe ASCI Red really was a VxWorks machine…

Doctor, It Hurts When I use Linux…

U.S. DEPARTMENT OF ENERGY

The rate and effect of undetected (aka silent) errors must be better understood.

- During acceptance, RR experienced intermittent, but relatively frequent (20 microhertz) silent errors on HPL
- The issue was eventually tracked to an entire MPI transfer filled with zeroes
  - But data on the sending side was confirmed to be correct
- Root cause was a policy misunderstanding between
  - System: *when I move pinned memory, I will tell you*
  - MPI: *you won't move pinned memory, so I won't listen*



*Slide courtesy of Andy White (LANL)

OS/R is Really a Set of APIs

- glibc and toolchain is what most application developers care about
  - Lightweight kernels can be Linux API and ABI compatible
- System programmers care about the OS
  - Tool developers drive the need for OS functionality more than applications
    - ptrace and signals are not ideal
- Observing application experience with accelerators is interesting
  - Proprietary hardware
  - Custom programming language
  - Cross-compile environment
  - Limited debugging support
  - Explicit memory management
  - No system calls
  - Dealing with a lightweight kernel should be easy after programming for accelerators

What's Driving the Need for More Advanced Runtime Systems?

- Dynamic local resource management
  - Massive on-node parallelism
    - Large numbers of threads that must be created, synchronized, and destroyed
  - Resilience
    - Node-level resources may come and go
  - Locality management
    - Reduce data movement to manage power
    - Potentially moving work to data
  - Scalability
    - Need to move away from bulk synchronous approach
    - Jitter will be pervasive
  - Hybrid programming models
    - Interoperability between different models
      - Distributed memory, shared memory, heterogeneous cores
    - Efficient phase change
      - Managing resources when moving between models
- Responding to non-local events
  - Resilience
    - System-level resources may come and go

Co-Design

Co-design is a key element of the Exascale strategy

- Architectures are undergoing a major change
  - Single thread performance is remaining relatively constant and on chip parallelism is increasing rapidly
  - Hierarchical parallelism, heterogeneity
  - Massive multithreading
  - NVRAM for caching I/O
- Applications will need to change in response to architectural changes
  - Manage locality and extreme scalability (billion-way parallelism)
  - Potentially tolerate latency
  - Resilience?
- Unprecedented opportunity for applications/algorithms to influence architectures, system software and the next programming model
  - Hardware R&D is needed to reach exascale
- We will not be able to solve all of the exascale problems through architectures work only
- Co-design has become a buzzword for identifying challenges

Fundamental Capabilities for Co-Design

- Software agility
  - Applications
    - Need to identify an important, representative subset
    - Application code must be small and malleable
  - System software
    - Smaller is better
    - Lightweight is ideal
    - Toolchain is always a huge issue
- Hardware simulation tools
  - Sandia SST
  - Virtualization
    - Leverage virtual machine capability to emulate new hardware capability
- Need mechanisms to know the impact of co-design quickly
- Integrated teams
  - Co-design centers

Hardware Support for Run-Time Systems

- Network hardware support for thread activation
  - Run-time system components must communicate across nodes
  - Message reception in current networks occurs by recognizing change in memory
    - Leads to polling
  - Need hardware mechanism to block/unblock threads on network events
  - Active message model only makes sense with hardware support
    - Waiting until there's nothing to do to notice incoming messages is bad
- More advanced network functions (eureka, dynamic hierarchy)
- More sophisticated mode switch / protection hardware
- Hardware performance information
  - Dynamic resource management decisions will need performance info
  - Current performance counters only capture a subset of what is needed
- Thread scheduling
  - Hardware support for efficient scheduling and synchronization
  - Must be flexible (programmable?)
  - Should allow for operating on groups of threads

Processor Protection Rings

- Current scalable HPC applications don't make system calls
  - Allows the ratio of full-featured service nodes to lightweight nodes to be small
  - All "real" system calls on Sandia LWK were serialized through one process
- Current run-time systems don't make system calls either
  - Only at set-up and tear-down
- Probably only need a small subset of cores with ring 0 capability
  - System calls will turn into run-time thread activation response
- May need to have more sophisticated network protection mechanism
  - Would like to have run-time system threads invoked on message arrival

Limited Coupling at OS Layer

- This is part of what defines the OS and differentiates run-time system
  - The lowest level of local hardware management
- Need hierarchical structure to allow for scalability
- Exascale will require tighter coupling between some components
  - Runtime system components
  - RAS system and runtime system
  - Application and runtime system
- Need to provide information while minimizing dependencies
  - Use all information but limit required information
  - OS shouldn't require non-local information

Acknowledgments

(People from whom I stole slides and/or ideas)

- Barney Maccabe (ORNL)
- Kevin Pedretti (SNL)
- Rolf Riesen (IBM)
- Marc Snir (UIUC)
- Andy White (LANL)