SAND2012-4725P

Exceptional service in the national interest

Sandia National Laboratories

System Software Challenges for Exascale Systems

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.

Outline

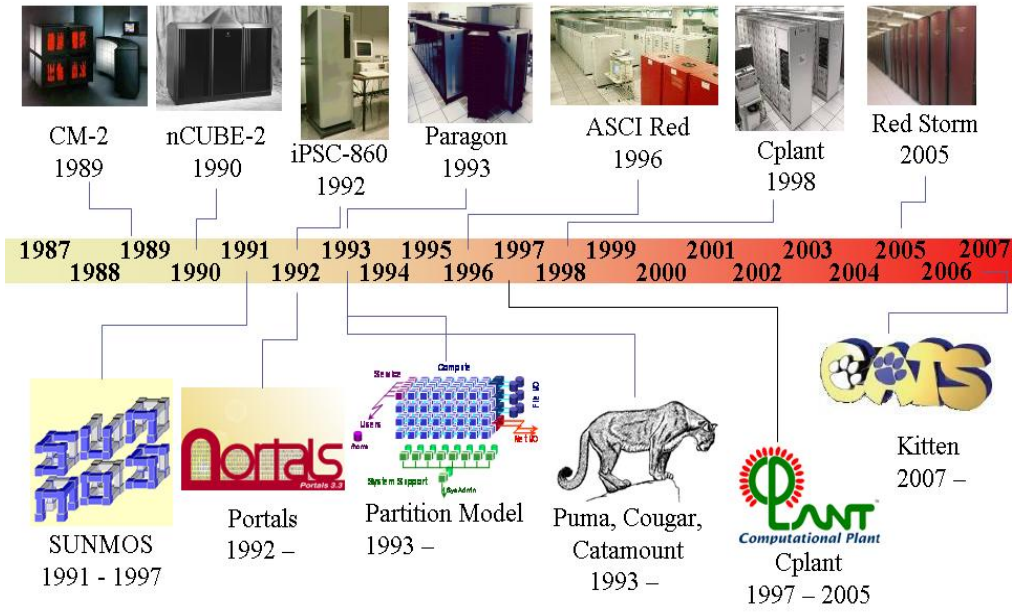
- Sandia system software R&D history
- Exascale computing
- System software challenges
- High-performance networking challenges
- Portals networking programming interface

System Software@Sandia

- Established the functional partition model for HPC systems
  - Tailor system software to function (compute, I/O, user services, etc.)
- Pioneered the research, development, and use of lightweight kernel operating systems for HPC
  - Only DOE lab to deploy OS-level software on large-scale production machines
  - Provided blueprint for IBM BlueGene OS
- Set the standard for scalable parallel runtime systems for HPC
  - Fast application launch on tens of thousands of processors
- Significant impact in the design and of scalable HPC interconnect APIs
  - Only DOE lab to deploy low-level interconnect API on large-scale production machines

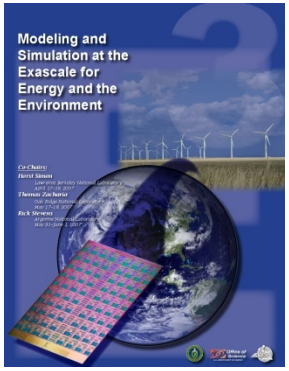
AWARDS:

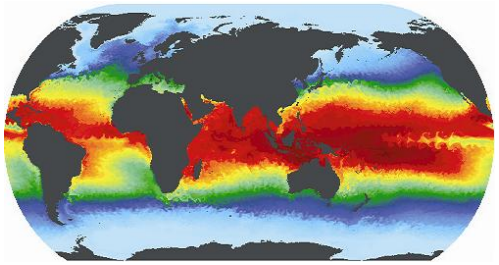
- **1998** Sandia Meritorious Achievement Award, TeraFLOP Computer Installation Team
- **2006** Sandia Meritorious Achievement Award, Red Storm Design, Development and Deployment Team
- **2006** NOVA Award Red Storm Design and Development Team
- **2009** R&D 100 Award for Catamount N-Way Lightweight Kernel
- **2010** Excellence in Technology Transfer Award, Federal Laboratory Consortium for Technology Transfer
- **2010** National Nuclear Security Administration Defense Programs Award of Excellence

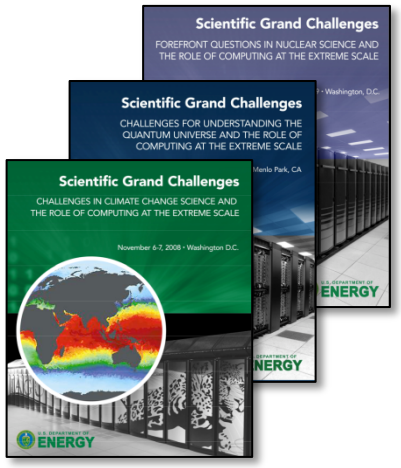


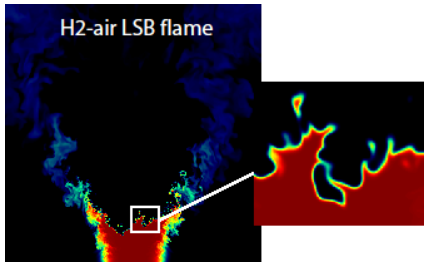
Process for identifying exascale applications and technology for DOE missions ensures broad community input

- Town Hall Meetings April-June 2007
- Scientific Grand Challenges Workshops Nov, 2008 – Oct, 2009
  - Climate Science (11/08),
  - High Energy Physics (12/08),
  - Nuclear Physics (1/09),
  - Fusion Energy (3/09),
  - Nuclear Energy (5/09),
  - Biology (8/09),
  - Material Science and Chemistry (8/09),
  - National Security (10/09)
  - Cross-cutting technologies (2/10)
- Exascale Steering Committee
  - "Denver" vendor NDA visits 8/2009
  - SC09 vendor feedback meetings
  - Extreme Architecture and Technology Workshop 12/2009
- International Exascale Software Project
  - Santa Fe, NM 4/2009; Paris, France 6/2009; Tsukuba, Japan 10/2009

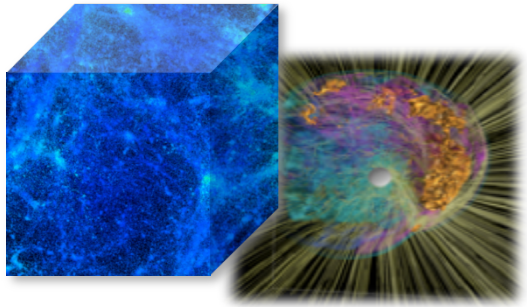








MISSION IMPERATIVES



FUNDAMENTAL SCIENCE

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

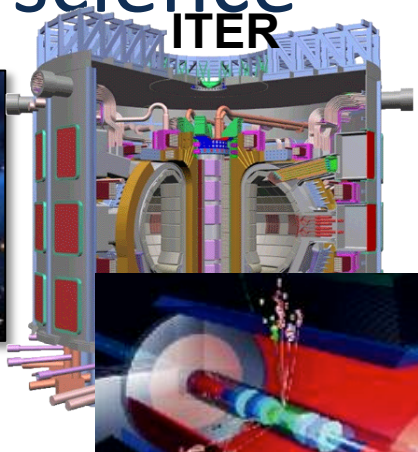
Accomplishing these missions requires exascale resources.

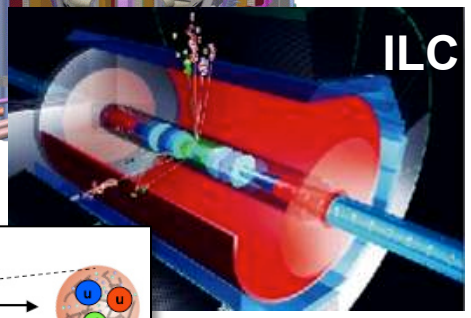
Exascale simulation will enable fundamental advances in basic science

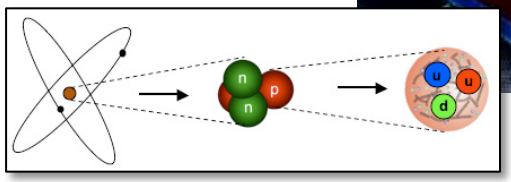
- High Energy & Nuclear Physics
  - Dark-energy and dark matter
  - Fundamentals of fission fusion reactions
- Facility and experimental design
  - Effective design of accelerators
  - Probes of dark energy and dark matter
  - ITER shot planning and device control
- Materials / Chemistry
  - Predictive multi-scale materials modeling: observation to control
  - Effective, commercial technologies in renewable energy, catalysts, batteries and combustion
- Life Sciences
  - Better biofuels
  - Sequence to structure to function

These breakthrough scientific discoveries and facilities require exascale applications and resources.

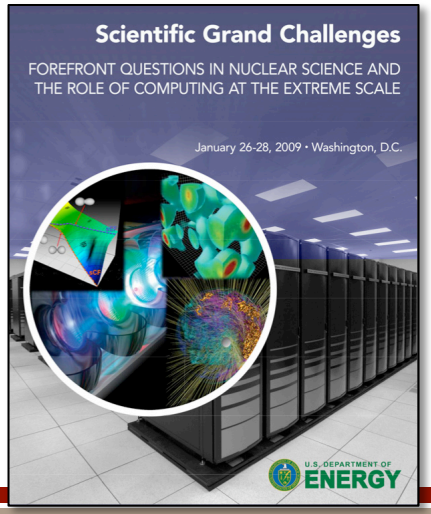
Hubble image of lensing

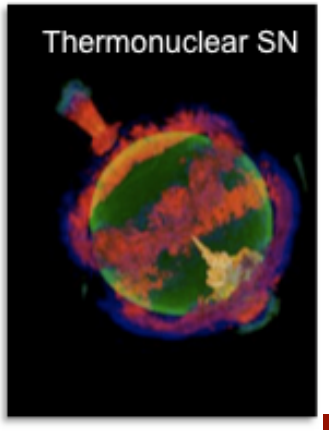






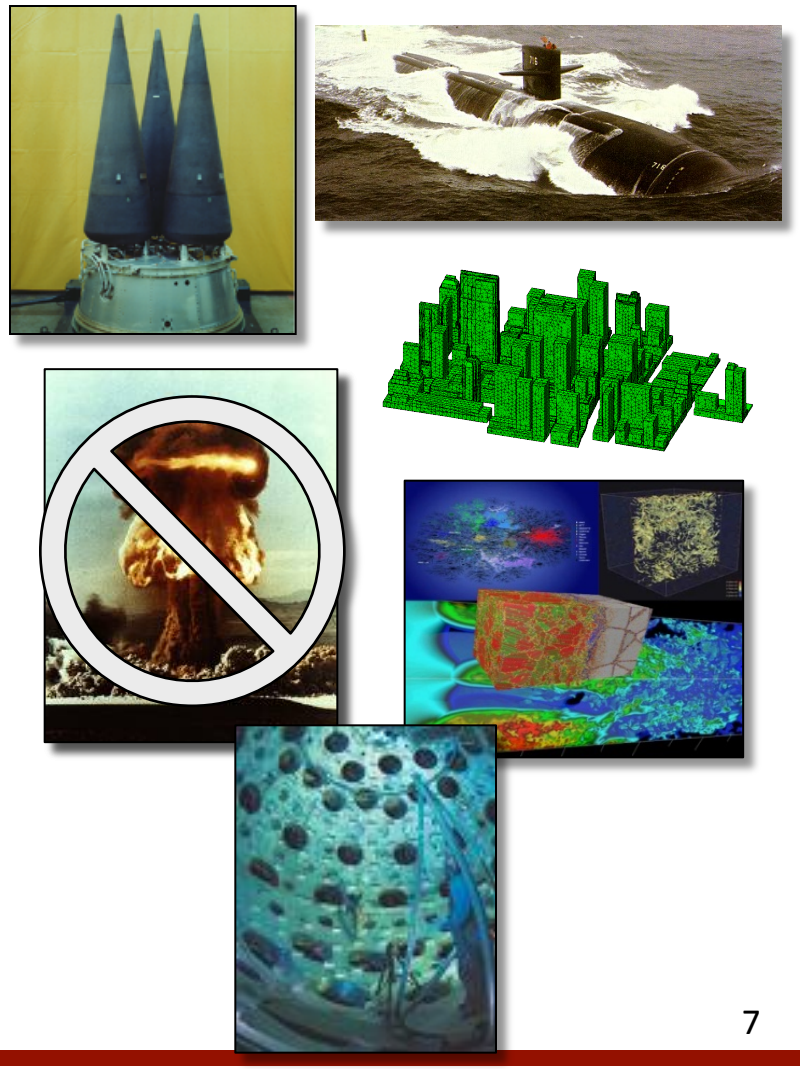
Structure of nucleons



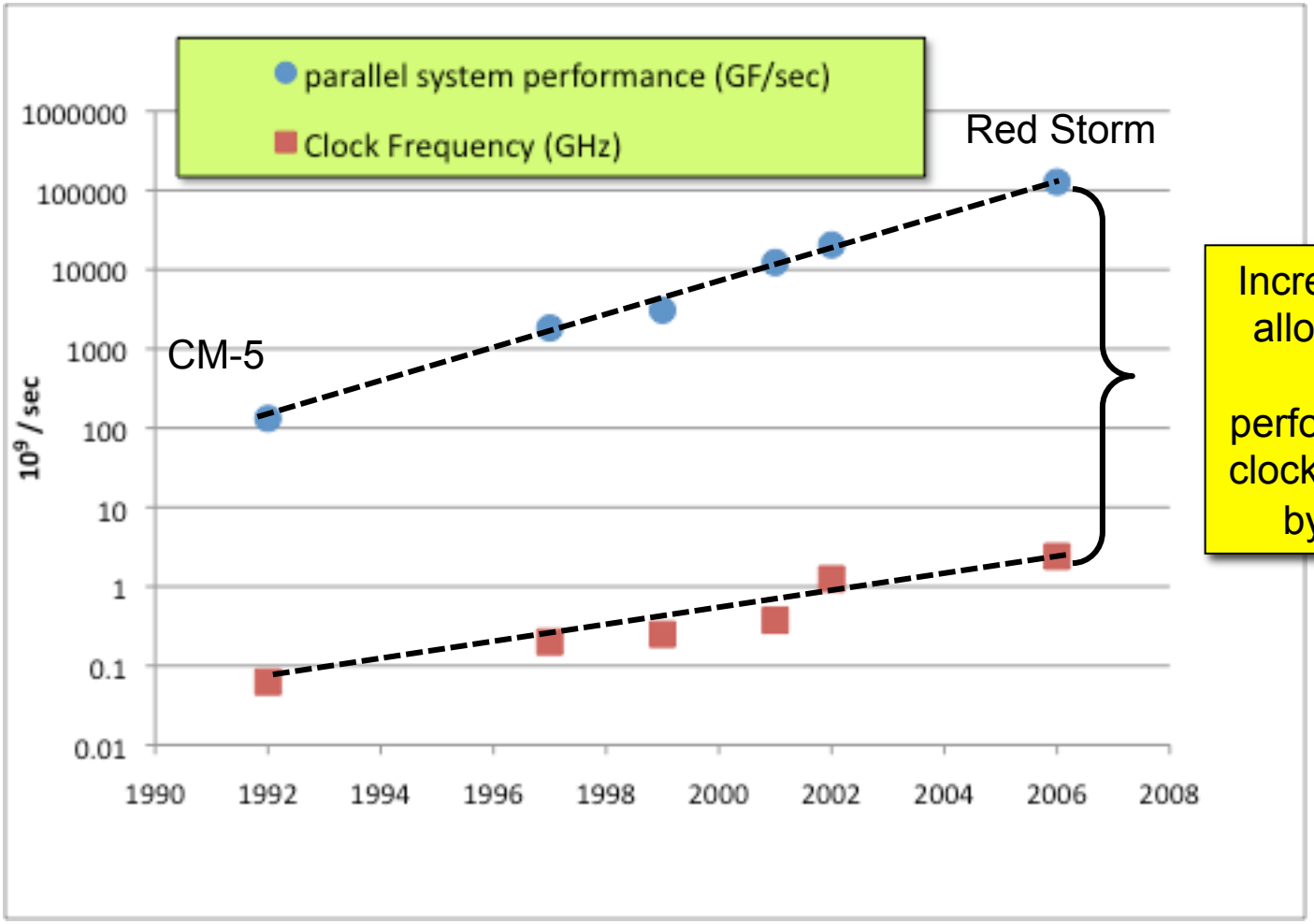


National Nuclear Security

- U.S. Stockpile must remain safe, secure and reliable without nuclear testing
  - Annual certification
  - Directed Stockpile Work
  - Life Extension Programs
- A predictive simulation capability is essential to achieving this mission
  - Integrated design capability
  - Resolution of remaining unknowns
    - Energy balance
    - Boost
    - Si radiation damage
    - Secondary performance
  - Uncertainty Quantification
  - Experimental campaigns provide critical data for V&V (NIF, DARHT, MaRIE)
- Effective exascale resources are necessary for prediction and quantification of uncertainty



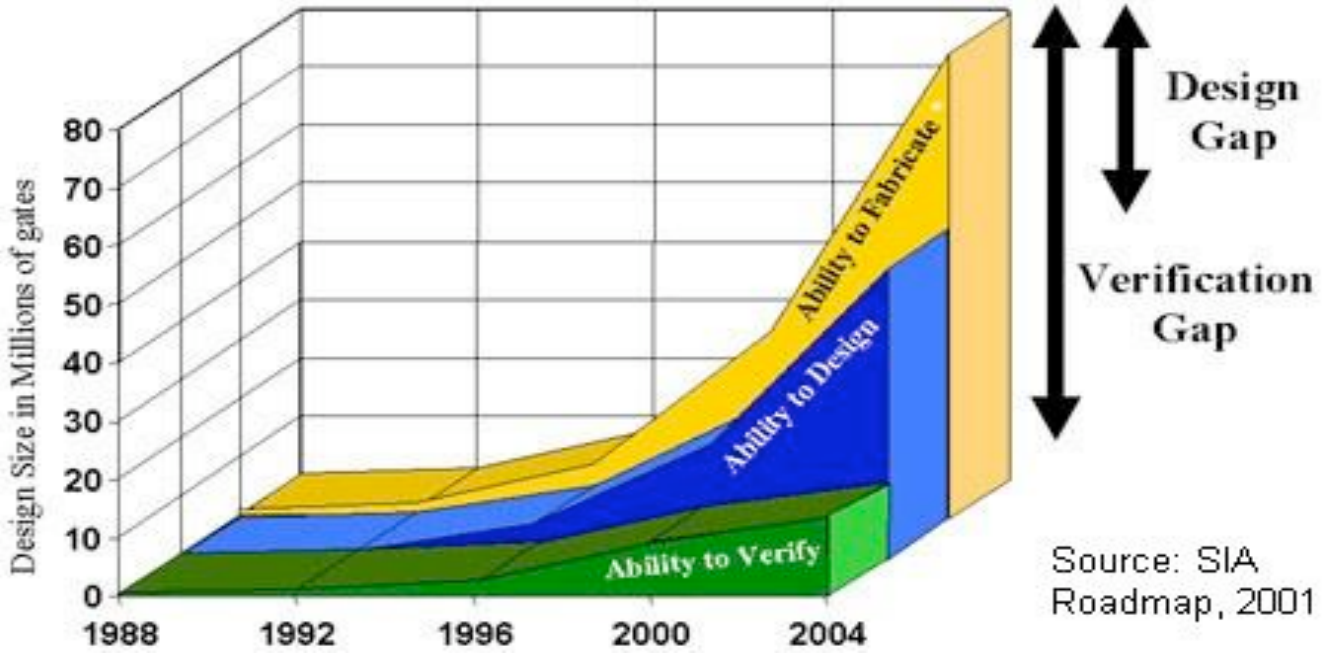
Concurrency is one key ingredient in getting to exaflop/sec



Increased parallelism allowed a 1000-fold increase in performance while the clock speed increased by a factor of 40

and power, resiliency, programming models, memory bandwidth, I/O, …

Many-core chip architectures are the future.



The shift toward increasing parallelism is not a triumphant stride forward based on breakthroughs in novel software and architectures for parallelism … instead it is actually a retreat from even greater challenges that thwart efficient silicon implementation of traditional uniprocessor architectures.
Kurt Keutzer

What are critical exascale technology investments?

- 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.
- Programming model. 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.

Potential System Architecture Targets

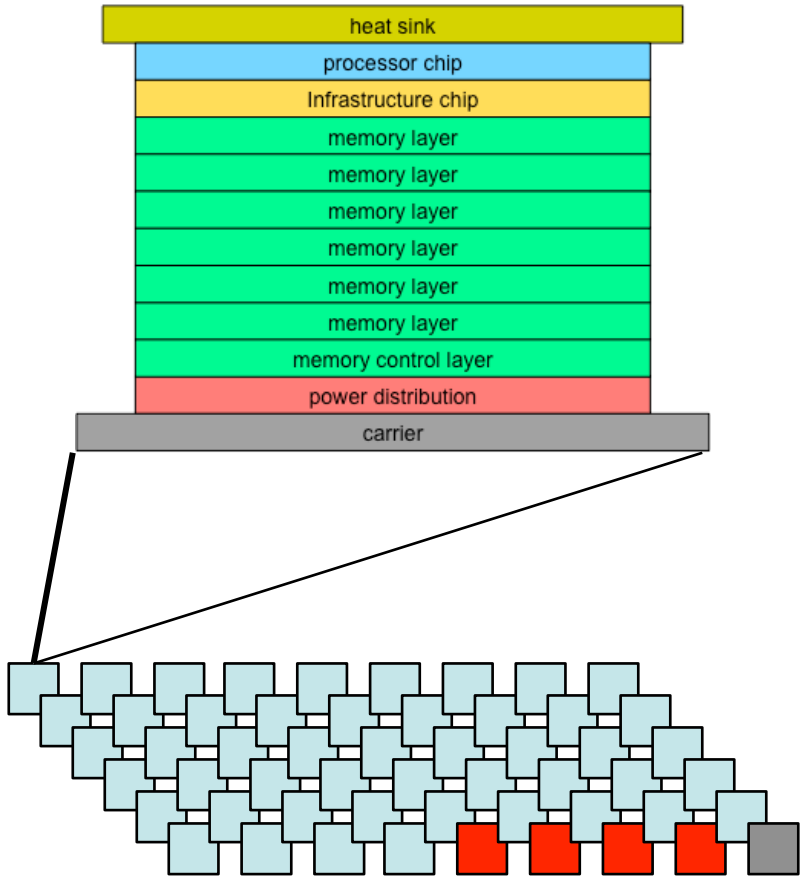
System attributes	2010	"2015"		"2018"	
System peak	2 Peta	200 Petaflop/sec		1 Exaflop/sec	
Power	6 MW	15 MW		20 MW	
System memory	0.3 PB	5 PB		32-64 PB	
Node performance	125 GF	0.5 TF	7 TF	1 TF	10 TF
Node memory BW	25 GB/s	0.1 TB/sec	1 TB/sec	0.4 TB/sec	4 TB/sec
Node concurrency	12	O(100)	O(1,000)	O(1,000)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

Factors driving up the fault rate

- Number of components both memory and processors will increase by an order of magnitude which will increase hard and soft errors.
- Smaller circuit sizes, running at lower voltages to reduce power consumption, increases the probability of switches flipping spontaneously due to thermal and voltage variations as well as radiation, increasing soft errors
- Power management cycling significantly decreases the components lifetimes due to thermal and mechanical stresses.
- Resistance to add additional HW detection and recovery logic right on the chips to detect silent errors. Because it will increase power consumption by 15% and increase the chip costs.
- Heterogeneous systems make error detection and recovery even harder, for example, detecting and recovering from an error in a GPU can involve hundreds of threads simultaneously on the GPU and hundreds of cycles in drain pipelines to begin recovery.
- Increasing system and algorithm complexity makes improper interaction of separately designed and implemented components more likely.

Programming Model Approaches

- Hierarchical approach (intra-node + inter-node)
  - Part I: Inter-node model for communicating between nodes
    - MPI scaling to millions of nodes: Importance high; risk low
    - One-sided communication scaling: Importance medium; risk low
  - Part II: Intra-node model for on-chip concurrency
    - Overriding Risk: No single path for node architecture
    - OpenMP, Pthreads: High risk (may not be feasible with node architectures); high payoff (already in some applications)
    - New API, extended PGAS, or CUDA/OpenCL to handle hierarchies of memories and cores: Medium risk (reflects architecture directions); Medium payoff (reprogramming of node code)
- Unified approach: single high level model for entire system
  - High risk; high payoff for new codes, new application domains



System Software Challenges

- Power/energy as a new fundamental resource
  - Minimizing data movement is key
- Memory capacity
  - Likely the scarcest resource
- Resilience
  - Application as well as OS/Runtime
- Heterogeneity
  - Processors, memories, networks, etc.
- More dynamic
  - Introspective and introspectable
- Scalability
  - Managing millions of cores rather than thousands of cores

System Software Opportunities

- Need fundamental new interfaces between system software, applications, tools, and hardware
  - Power management, memory management, data movement, dynamic resource management, etc.
- Lightweight approaches that minimize memory use and reduce complexity will continue to be important
- Some explorations may need to be done without hardware
  - Simulation and emulation environments, virtualization, mini-applications, synthetic workloads will be needed

Simulation, emulation, and performance modeling

- Need to develop applications, algorithms, and software for hardware and systems that don't exist yet
- Exascale systems will be significantly different from today's systems
  - Massive amounts of on-node parallelism
  - Heterogeneity
  - More layers of memory hierarchy
- System simulation, emulation, performance models, and tools are all methods that have been employed to predict how algorithms and applications perform on future systems
  - Such activities are also useful for feedback to the architecture community
- Hardware/software co-simulation used extensively in embedded computing
  - Cycle-accurate application simulation at exascale is not feasible
  - Multi-level simulation tools may be useful
  - Cycle-accurate simulators or emulation might be used to predict node performance
  - Less complex models can also provide meaningful insight

Mini-Applications

- Proven to be useful for co-design as proxies of full applications
  - Assessing not only impact of low-level hardware, but also re-design of application or algorithm
- A well-designed mini-app
  - Contains one or more performance-impacting features
  - Has a small code base
  - Can be easily re-written
  - Is open source
- Working with mini-app elevates discussion between various system component developers
- Broadens the scope of contributors who may not have time or access to work with full application
- Mini-apps are expected to lose relevance over time
- Research opportunities
  - Automated methods of generating mini-apps
  - Determining the most computationally intensive portions of a code on future systems

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 can be 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

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

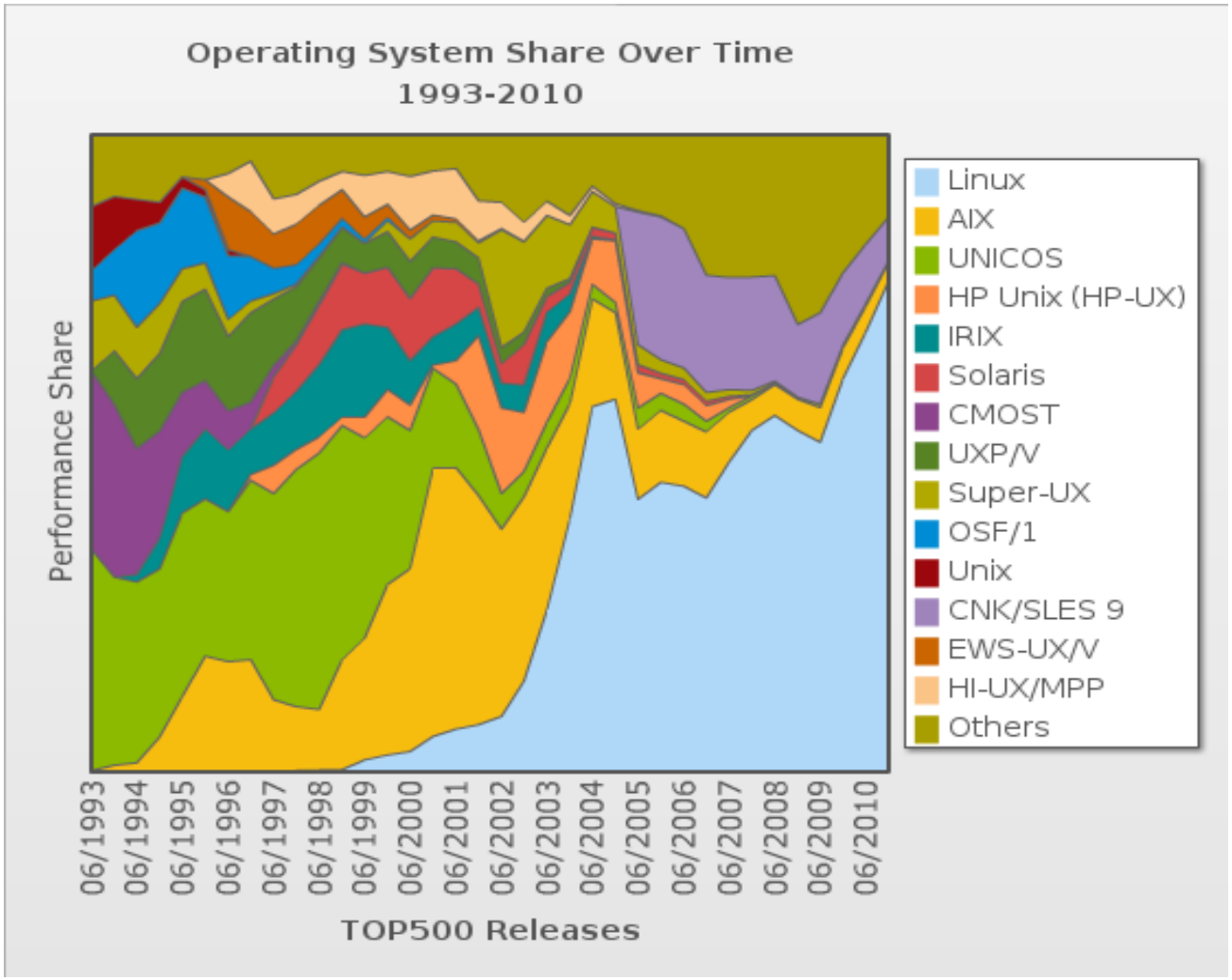
Driving the Need for More Advanced Runtime Systems

- 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
    - Potentially recomputing
  - 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

OS/R Issues and Challenges

- Fault tolerance / resilience
- Programming models
- OS structure
- APIs
- Specific functionality
- Scalability
- Interactivity
- Future hardware
- Hardware support for Oses
- Application requirements
- Metrics
- Programmatic challenges
- Heterogeneity
- Degree of transparency
- Infrastructure support for multiple OS/Rs
- Vendor proprietary components
- Tools support/requirements
- Desktop integration
- Dynamic resource management
- Vendors
- Testbeds
- Adaptation
- Usage models
- Memory hierarchy
- Security
- Standards
- Portability
- Culture
- Non-traditional architectures
- Multiple management policies
- Mainstream technology overlap
- Support for introspection
- Interface to RAS
- Testing
- Application requirements
- Intellectual property
- Sustainability
- Energy/power

Linux is the Dominant OS on the Top 500



November 2011 Top 500 Operating Systems

Are These Really Linux Supercomputers?

- #2 - Tianhe-1A
  - 14,336 6-core Intel Xeons
    - 86,016
    - 3%
  - 7168 448-core Nvidia GPUs
    - 3,211,264 total cores
    - 97%
- #10 - 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…

Challenge areas for HPC networks

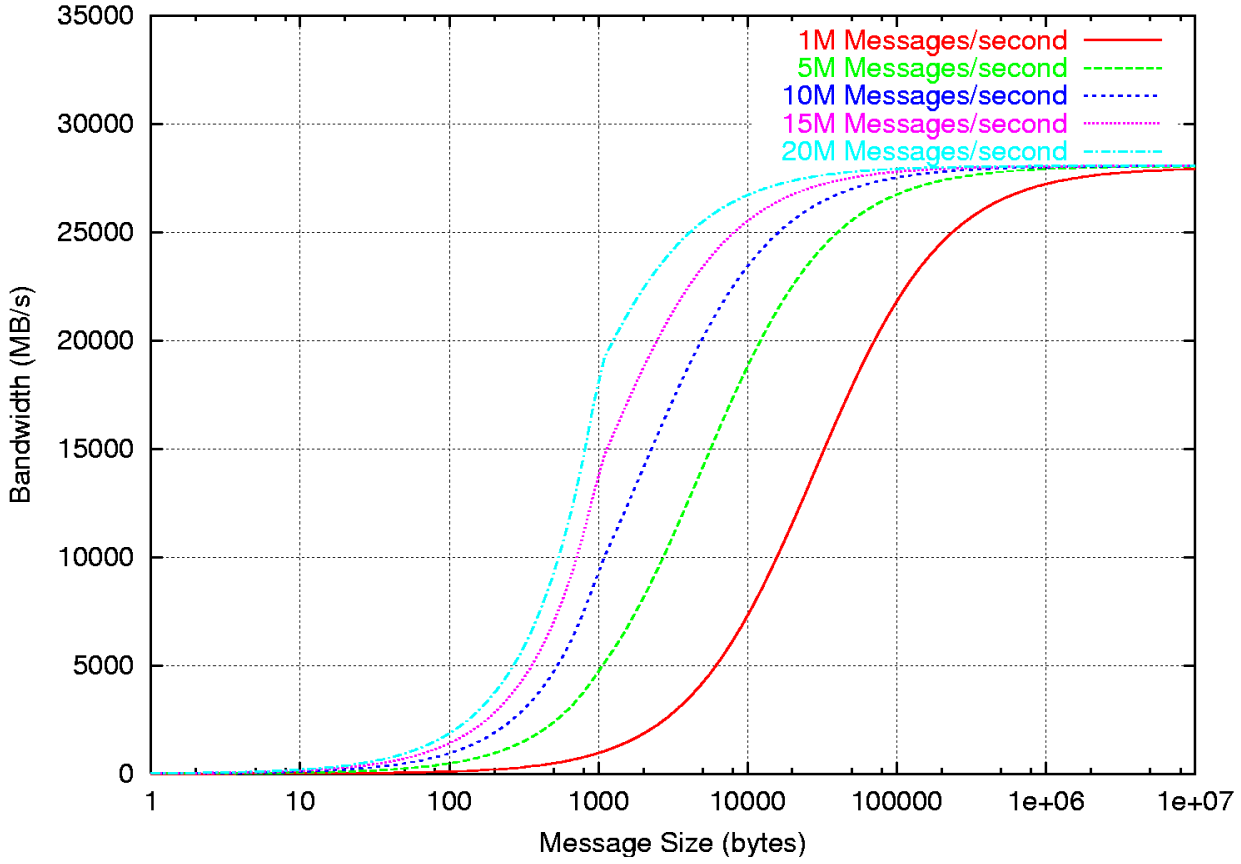
- The traditional “big three”
  - Bandwidth
  - Latency
  - Message Rate (Throughput)

- Other important areas for “real applications” versus benchmarks
  - Allowable outstanding messages
  - Host memory bandwidth usage
  - Noise (threading, cache effects)
  - Synchronization
  - Progress
  - Topology
  - Reliability

MPI Will Likely Persist Into Exascale Era

- Number of network endpoints will increase significantly (5-50x)
- Memory and power will be dominant resources to manage
  - Networks must be power and energy efficient
  - Data movement must be carefully managed
  - Memory copies will be very expensive
- Impact of unexpected messages must be minimized
  - Eager protocol for short messages leads to receive-side buffering
  - Need strategies for managing host buffer resources
  - Flow control will be critical
  - N-to-1 communication patterns will (continue to be) disastrous
- Must preserve key network performance characteristics
  - Latency
  - Bandwidth
  - Message rate (throughput)

High Message Throughput is Vital



Message rate determines the minimum message size needed to saturate the available network bandwidth

Current flow control strategies are not sufficient

- Credit-based
  - Limit number of outstanding send operations
  - Used credits are replenished implicitly or explicitly
  - Effectiveness limited to N-to-1 scenario
  - Potential performance penalty for well-behaved applications
- Acknowledgment-based
  - Receiver explicitly confirms receipt of every message
  - Significant per-message performance penalty
    - Round trip acknowledgment doubles latency
  - Performance penalty for well-behaved applications
- Local copying (bounce buffer) mitigates latency penalty
- Both strategies limit message rate and effective bandwidth
- Flow control implemented at user-level inside MPI library
- Network transport usually has its own flow control mechanism
  - No mechanism for back pressure from host resources to network

Applications must become more asynchronous

- Applications cannot continue to be bulk synchronous
  - Overhead of global synchronization will limit scaling
  - Global synchronization increases susceptibility to noise
- One-sided communication requires explicit synchronization
- Network API must provide asynchronous operations and progress
  - Data movement must be independent of host activity
- Active Messages
  - Polling is fundamental to all AM
  - Progress only when nothing else to do
  - Polling memory for message reception is inefficient
  - Needs hardware support to integrate message arrival with thread invocation
- Run-time systems will also need to communicate
  - Need to communicate evolving state of the system
  - Need a common portable API
  - Using TCP OOB connection will be infeasible

Resiliency will impact network API

- Network will need to expose errors to enable recovery
- Applications and system components will have different resiliency requirements
  - Reachability errors must be handled by run-time services
  - Graceful degradation may be appropriate for some applications
- May need OOB mechanism for recognizing network failures
  - AM or event-driven API would be ideal
  - Hardware support for network-level protection
    - RAS system invoking OS via network messages

PORTALS NETWORK PROGRAMMING INTERFACE

Portals Network Programming Interface

- Network API developed by Sandia, U. New Mexico, Intel
- Previous generations of Portals deployed on several production massively parallel systems
  - 1993: 1800-node Intel Paragon (SUNMOS)
  - 1997: 10,000-node Intel ASCI Red (Puma/Cougar)
  - 1999: 1800-node Cplant cluster (Linux)
  - 2005: 10,000-node Cray Sandia Red Storm (Catamount)
  - 2009: 18,688-node Cray XT5 – ORNL Jaguar (Linux)
- Focused on providing
  - Lightweight “connectionless” model for massively parallel systems
  - Low latency, high bandwidth
  - Independent progress
  - Overlap of computation and communication
  - Scalable buffering semantics
  - Protocol building blocks to support higher-level protocols
- Supports MPI, SHMEM, ARMCI, GASNet, Lustre, etc.

What makes Portals different?

- One-sided communication with optional matching
- Provides elementary building blocks for supporting higher-level protocols well
- Allows structures to be placed in user-space, kernel-space, or NIC-space
- Allows for zero-copy, OS-bypass, and application-bypass implementations
- Scalable buffering of MPI unexpected messages
- Supports multiple higher-level protocols within a process
- Run-time system independent
- Well-defined failure semantics

Portals 4.0:
Applying lessons learned from Cray SeaStar

- Allow for higher message rate
  - Atomic search and post for MPI receives required round-trip across PCI
  - Eliminate round-trip by having Portals manage unexpected messages
- Provide explicit flow control
  - Encourages well-behaved applications
    - Fail fast
    - Identify application scalability issues early
  - Resource exhaustion caused unrecoverable failure
  - Recovery doesn't have to be fast
  - Resource exhaustion will disable Portal
  - Subsequent messages will fail with event notification at initiator
  - Applies back pressure from network
    - Performance for scalable applications
    - Correctness for non-scalable applications

Portals 4.0 (cont'd)

- Enable a better hardware implementation
  - Designed for intelligent or programmable NICs
  - Arbitrary list insertion is bad
  - Remove unneeded symmetry on initiator and target objects
- New functionality for one-sided operations
  - Eliminate matching information
    - Smaller network header
    - Minimize processing at target
  - Scalable event delivery
    - Lightweight counting events
  - Triggered operations
    - Chain sequence of data movement operations
    - Asynchronous collective operations
      - Mitigate OS noise effects
    - Triggered rendezvous protocol
      - Enables progress without bandwidth penalty

Portals 4 Implementations

- OpenFabrics Verbs
  - Provided by System Fabric Works
  - Provides a high-performance reference implementation for experimentation
  - Help identify issues with API, semantics, performance, etc.
  - Independent analysis of the specification
- Shared memory
  - Offers consistent and understandable performance characteristics
  - Provides ability to accurately measure instruction count for Portals operations
  - Better characterization of operations that impact latency and message rate
  - Evaluation of single-core onloading performance limits
- Structural Simulation Toolkit
  - Partial implementation for exploring NIC structures for offload

Underwood, et al. “Enabling Flexible Collective Communication Offload with Triggered Operations,” in Proceedings of the IEEE Symposium on High-Performance Interconnects, August 2011.

TRIGGERED OPERATIONS FOR COLLECTIVE COMMUNICATION

Motivation

- Collectives are important to a broad array of applications
  - As node counts grow, it becomes hard to keep collective time low
- Offload provides a mechanism to reduce collective time
  - Eliminates portion of Host-to-NIC latency from the critical path
  - Relatively complex collective algorithms are constantly refined and tuned
- Building blocks provide a better
  - Allow algorithm research and implementation to occur on the host
  - Provides a simple set of hardware mechanisms to implement
- A general purpose API is needed to express the building blocks

Triggered Operations

- Lightweight events are counters of various network transactions
  - One counter can be attached to multiple different operations or even types of operations
  - Fine grained control of what you count is provided
- Portals operation is “triggered” when a counter reaches a threshold specified in the operations
  - Various types of operations can be triggered
  - Triggered counter update allows chaining of local operations

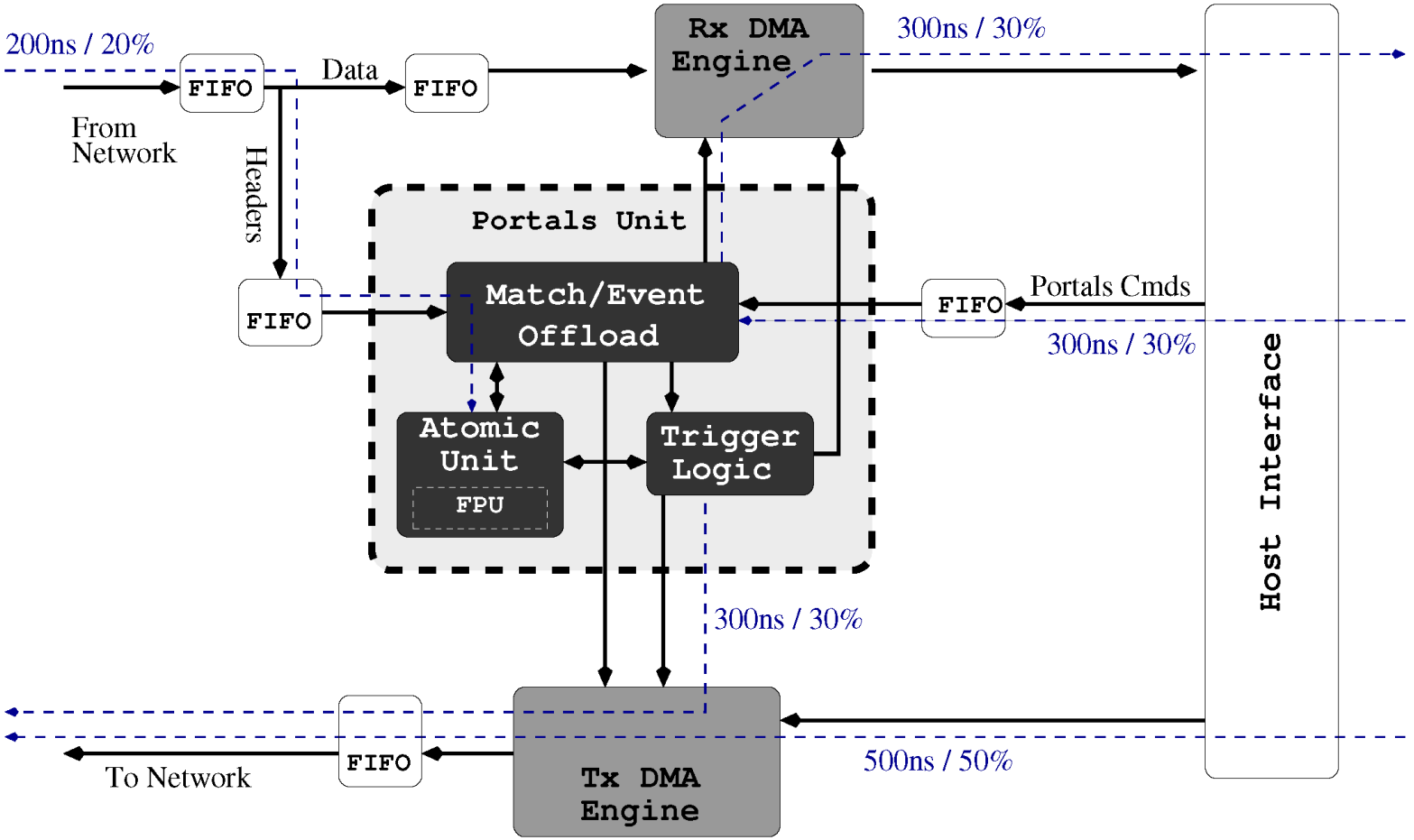
Generality of Triggered Operations

- Numerous collectives have been implemented so far
  - Allreduce
  - Bcast
  - Barrier
- Numerous algorithms have been implemented for multiple collectives
  - Binary tree
  - k-nomial tree
  - Pipelined broadcast
  - Dissemination barrier
  - Recursive doubling

Simulation Methodology

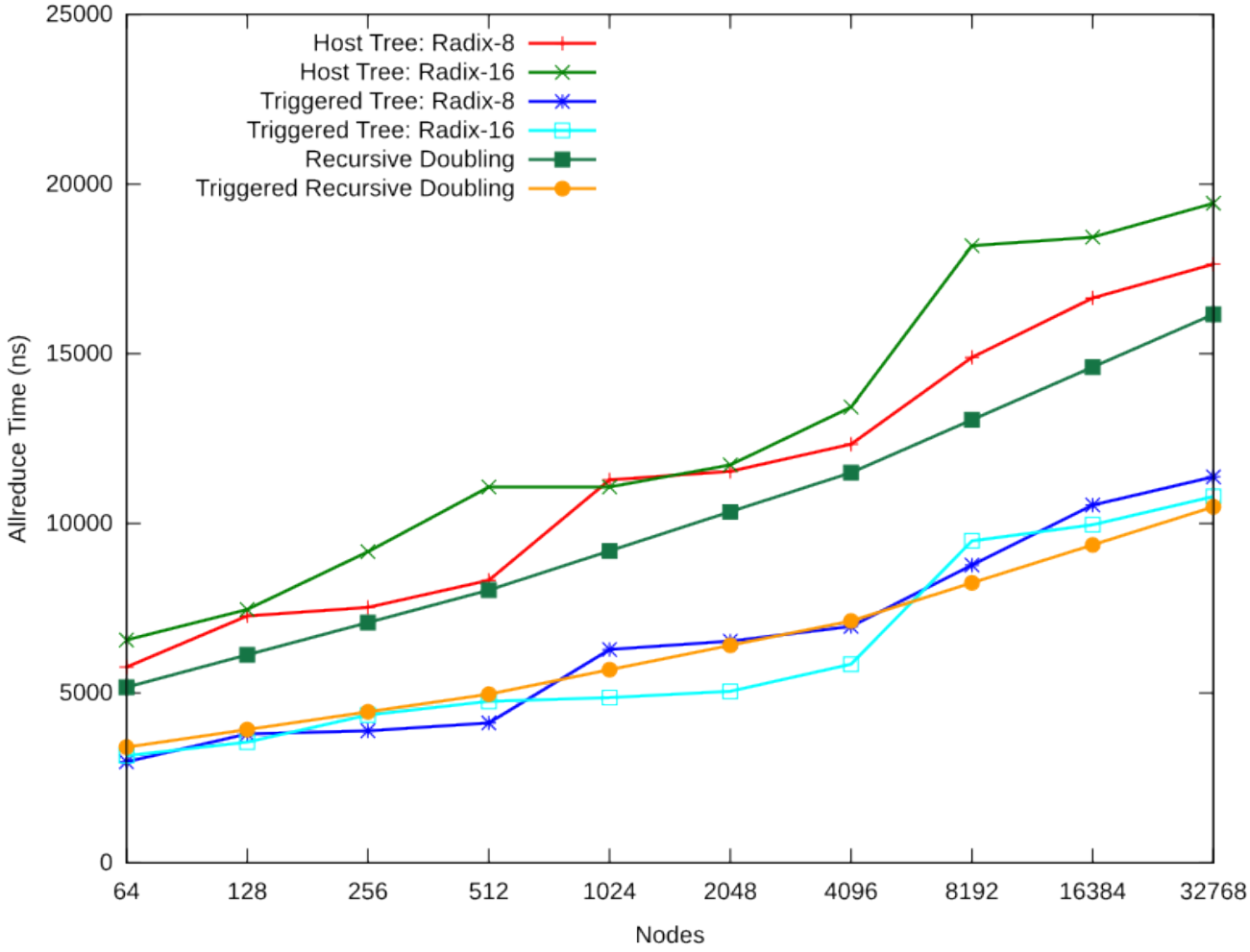
- Utilized SST simulator developed at Sandia
- Modeled processor and NIC as separate state machines
  - Fixed delays between states to model delays and overhead
  - Single state machine for processor, multiple for NIC to model concurrent hardware blocks
- Modeled several combinations of parameters defined by latency and message rate
  - Allocated delay to various units that were modeled

High-Level NIC Architecture



Allreduce

500ns, 10 Mmsgs/s

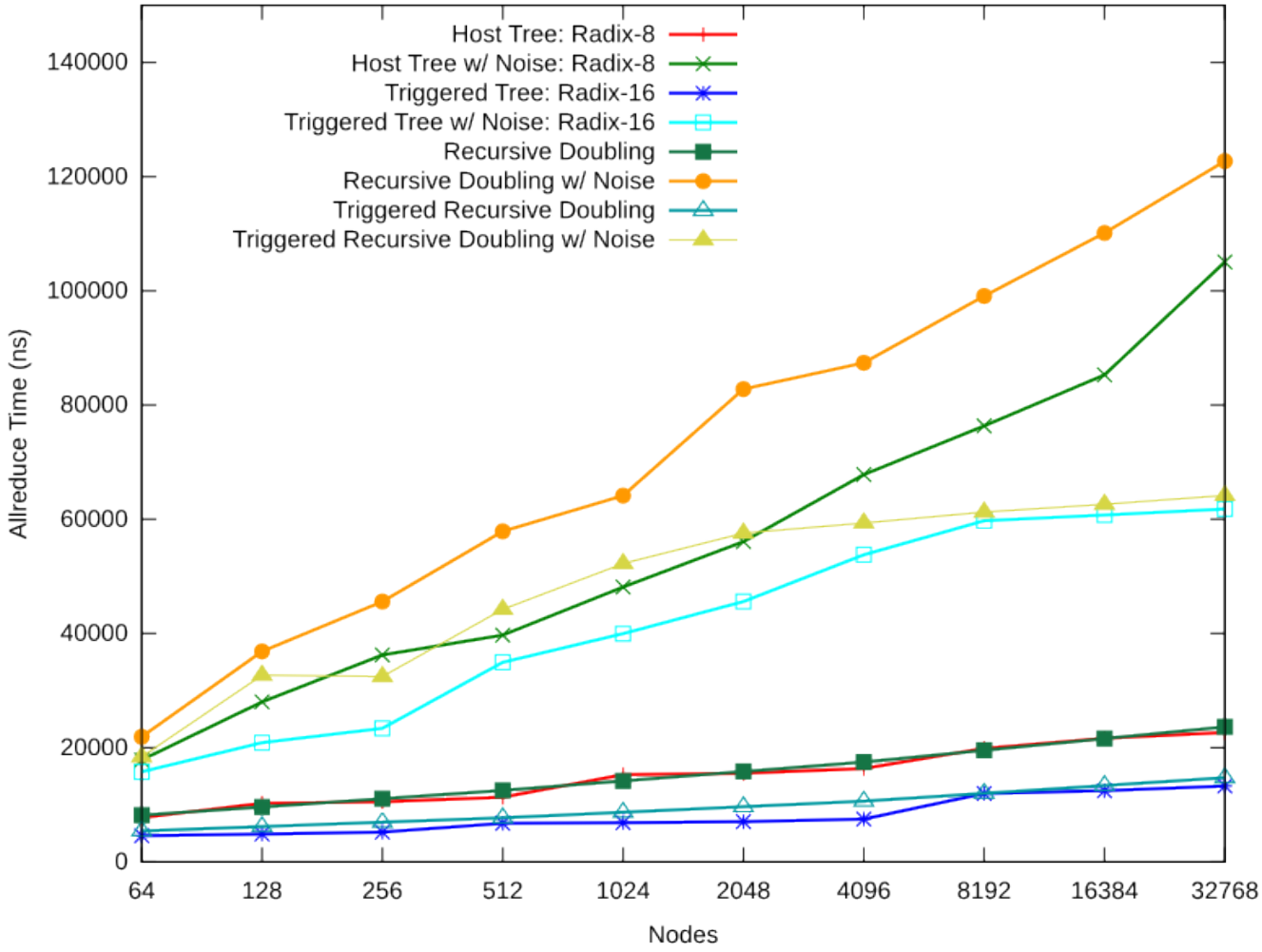


Noise Simulations

- Three noise profiles were simulated (2.5% noise for each)
  - 250 ns @ 100KHz
  - 25 μs @ 1KHz
  - 2.5 ms @ 10Hz
- Noise events were randomly distributed
  - Stopped all host processing during a noise event
  - NIC processing continued
- Timed individual collective operations (first entry to last exit)

Allreduce With Noise

25 us @ 1 KHz



Noise Simulation Results

- Recursive doubling has poor noise tolerance
- Offload gives significant improvement in noise tolerance
  - Partly from reduced time
  - Partly from reduced host participation
  - Synchronizing operation still cannot complete until everyone contributes a value
- Interesting shape of curves in middle noise case
  - Host based latency continues to grow with node count
  - NIC based latency plateaus

Interesting things we learned

- Time to initiate a transaction from the host to the NIC makes things difficult
  - Even with a high NIC rate, can be rate limited by the host
  - Limitation of using host to initiate all operations instead of offloading algorithm
  - If transactions are posted in correct order, limitation is effectively mitigated
- Proper message scheduling is important
  - Time between message initiations on the host (gap) matches network hop latency: send the far away ones first!
- k-nomial trees are better, but the work at the root limits the maximum value of k
- You can have speed or reproducibility, but…

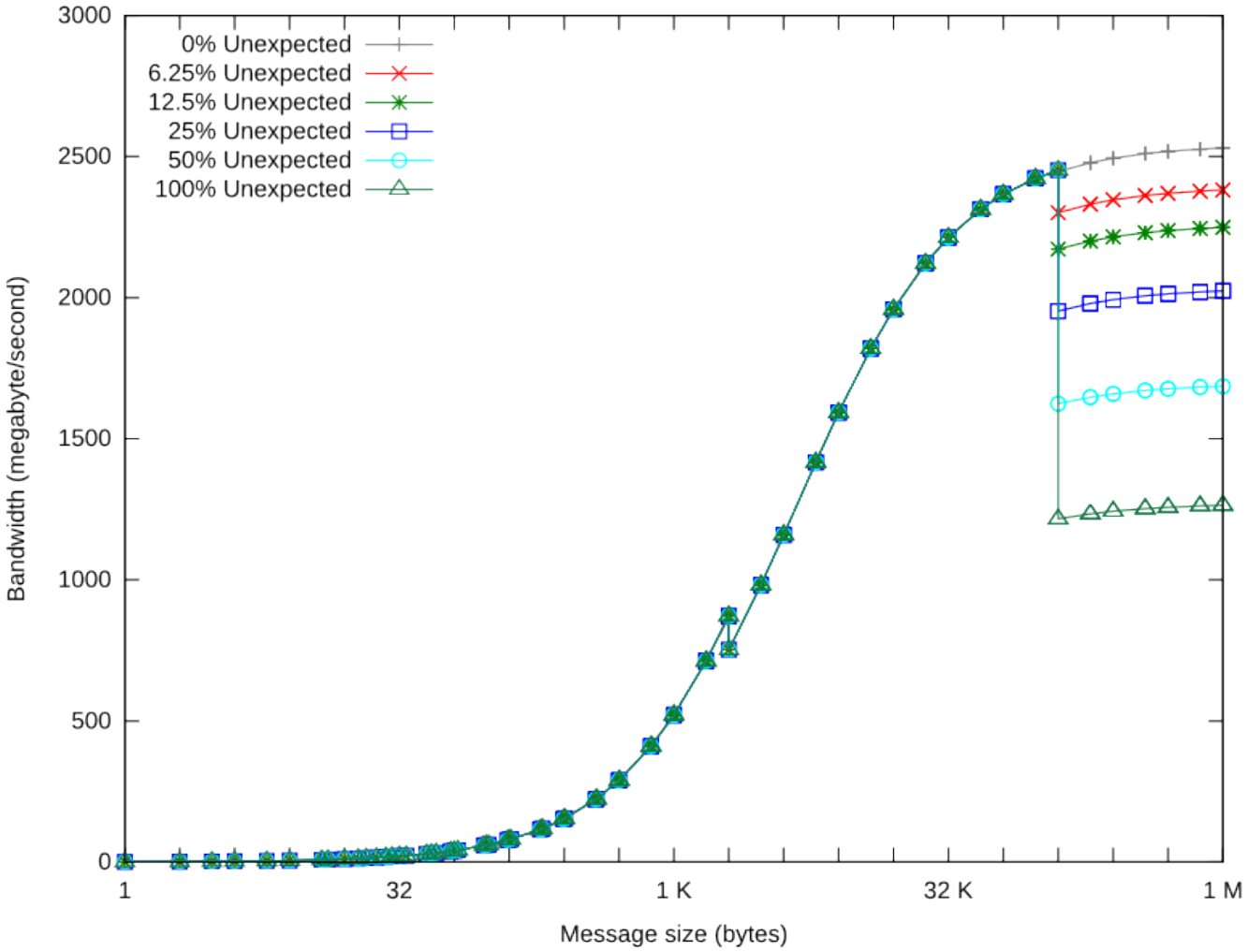
Triggered collectives summary

- Triggered operations provide a general set of building blocks
  - Supports a variety of collective operations
  - Supports a variety of algorithms
  - Has usage beyond just collectives offload
- Collective offload has limited performance upside versus idealized host implementation
  - 2x performance improvement due to improved latency and improved message rate
  - Performance could be improved somewhat by having host “push” data
- Noise sensitivity substantially reduced when operations are offloaded

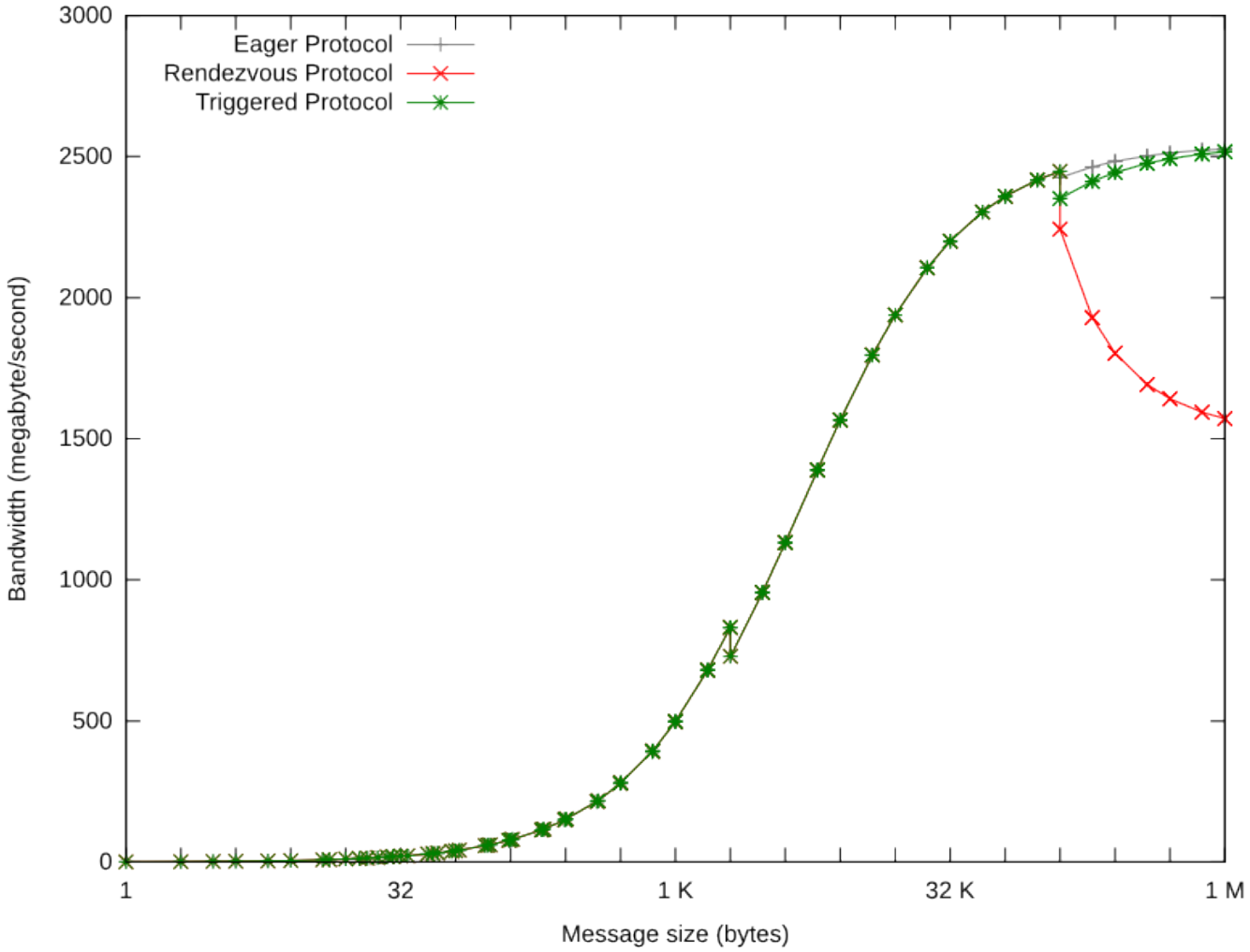
Barrett, Brightwell, Hemmert, Wheeler, Underwood. “Using Triggered Operations to Offload Rendezvous Messages,” in Proceedings of the European MPI Users’ Group Conference, September 2011.

TRIGGERED OPERATIONS FOR A RENDEZVOUS PROTOCOL

Ping-Pong Bandwidth



Ping-Pong Bandwidth

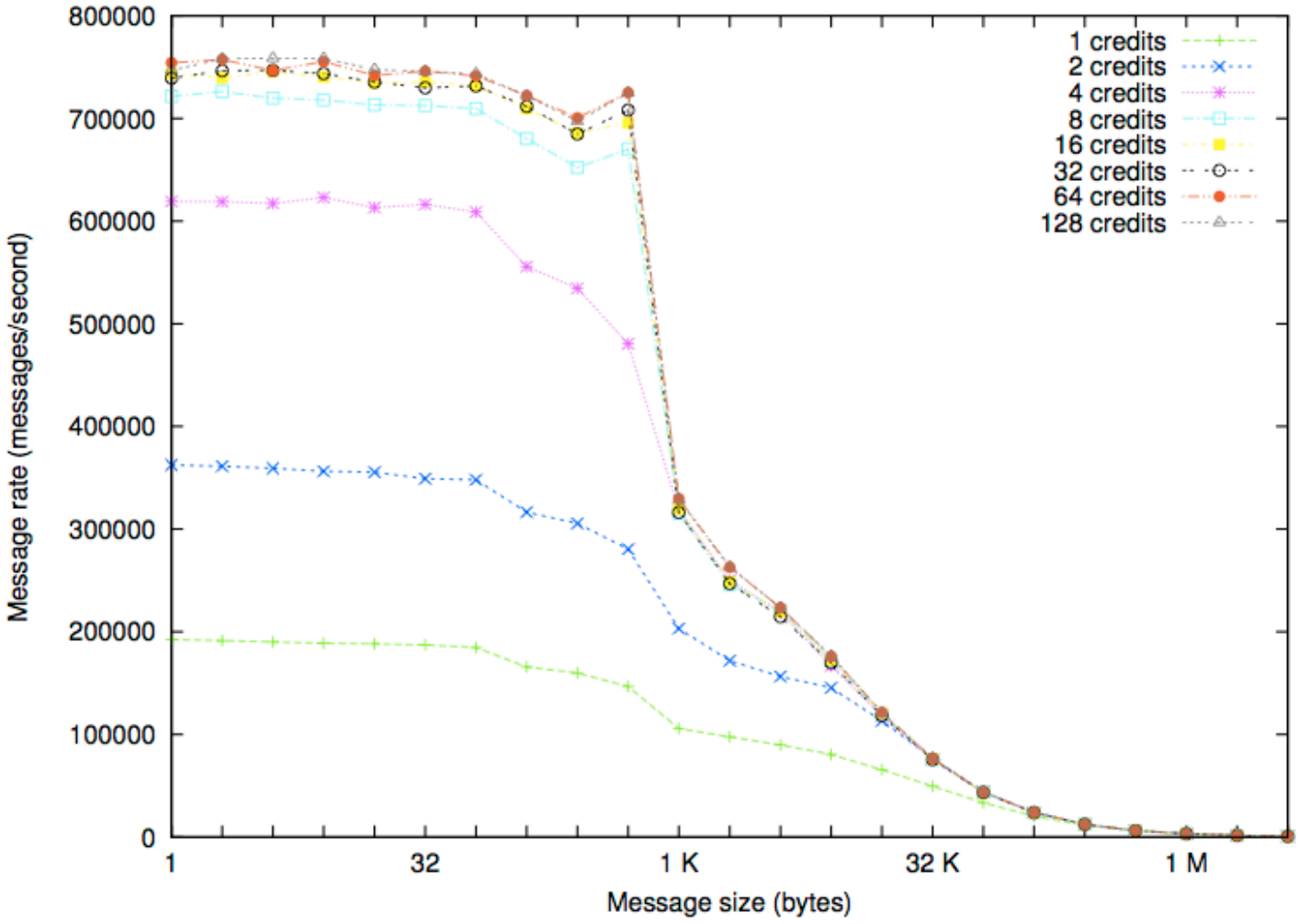


RECEIVER-MANAGED FLOW CONTROL

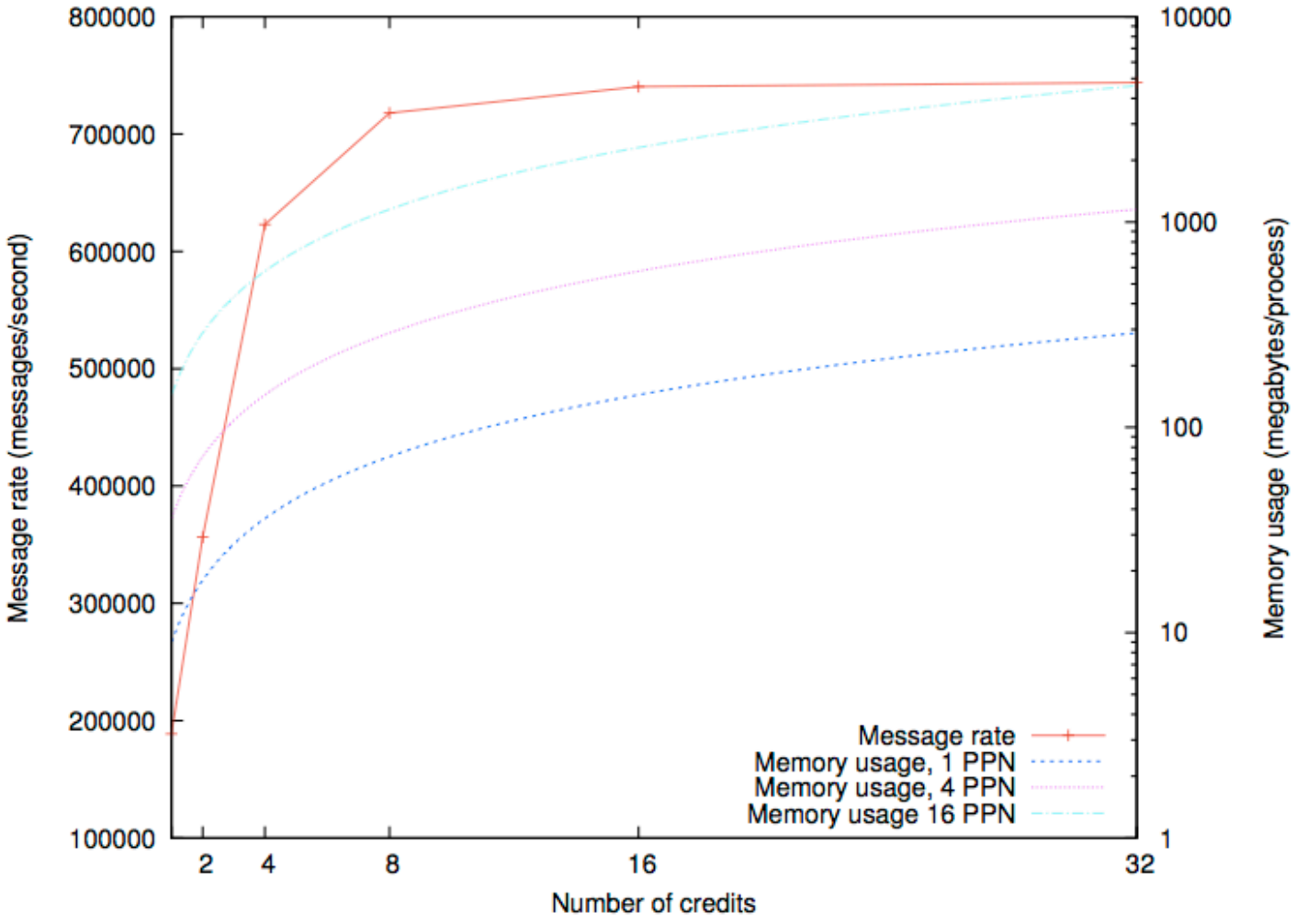
MPI Requires Flow Control

- MPI-1 Standard mandates that senders cannot overwhelm receivers with too many unexpected messages
- Two main strategies for providing flow control
  - Credit-based
    - A credit is needed to send a message
    - Credits given out at initialization or connection setup
    - Credits can be static or dynamic based on message intensity
    - Credits exchanged through explicit or piggyback messages
  - Acknowledgment-based
    - Wait for receiver to acknowledge message reception
    - ACKs can be explicit or piggyback messages
- Both strategies assume senders need to be constrained
- Our approach is to recover rather than constrain
  - Emphasize performance for well-designed applications
  - Provide correctness for poorly-designed applications

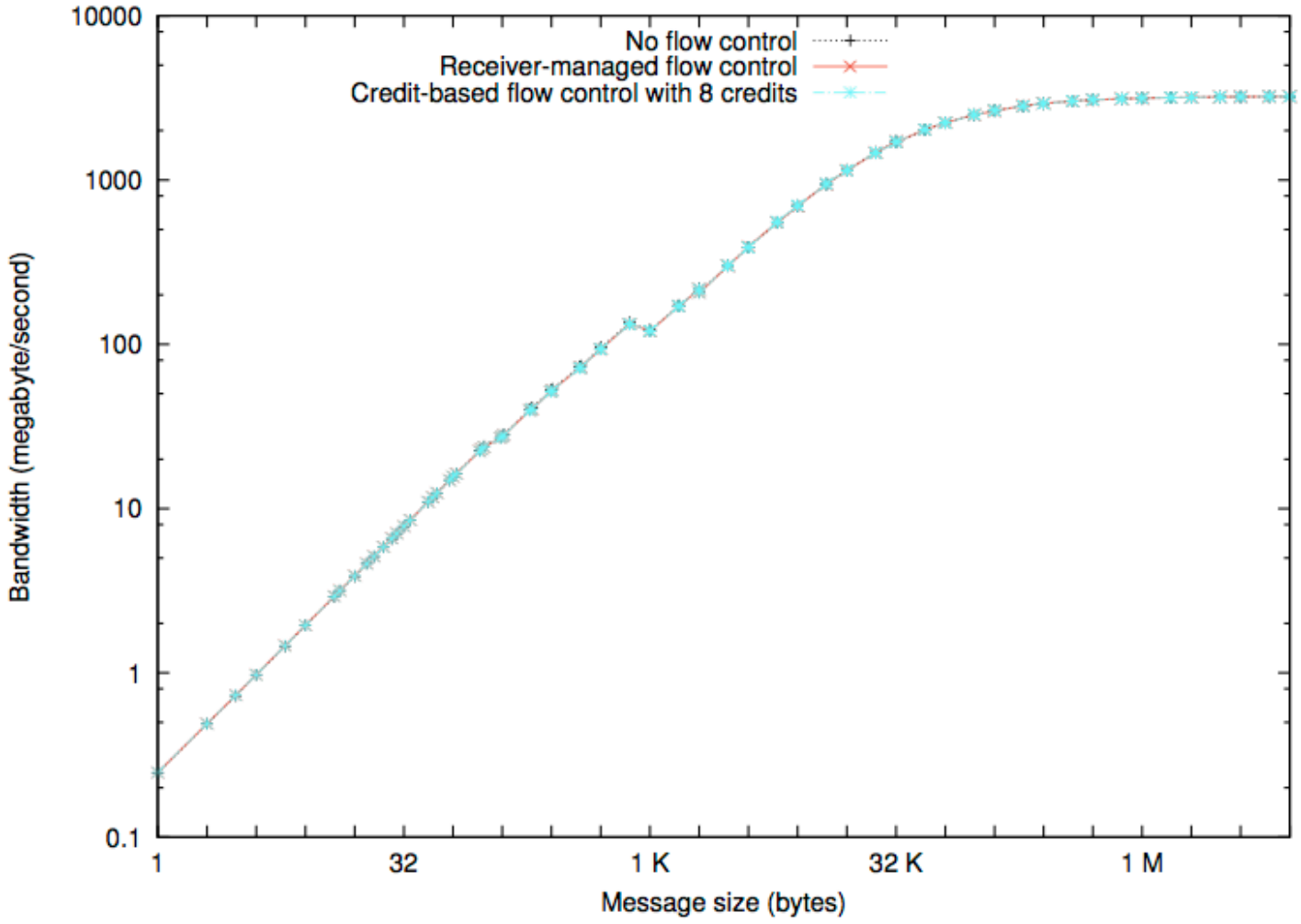
Too few credits can reduce message rate



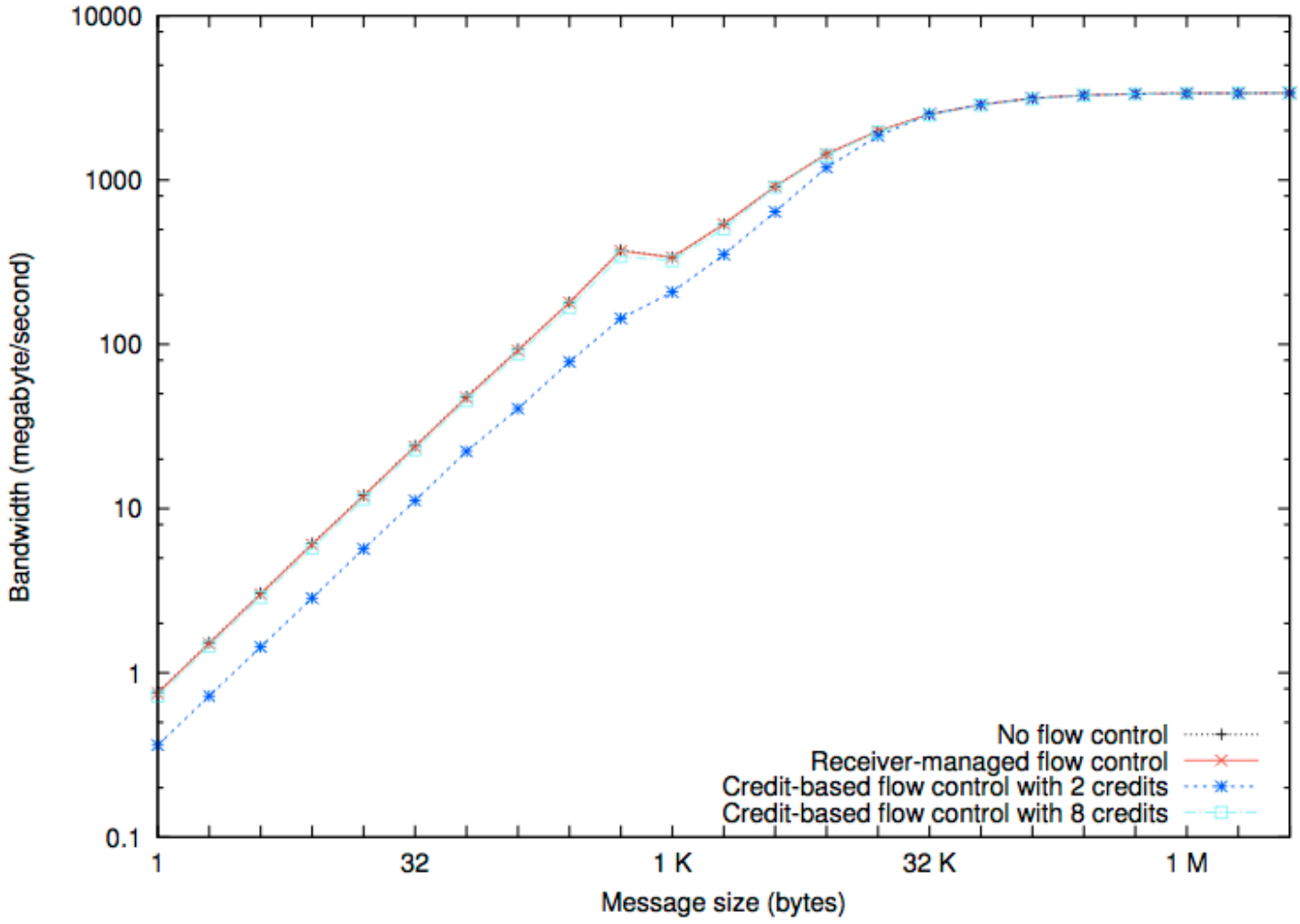
Too many credits wastes memory



Ping-pong bandwidth is unimpacted



Too few credits degrades streaming bandwidth



Acknowledgments

- DOE Exascale Initiative Steering Committee
- Sandia
  - Brian Barrett
  - Scott Hemmert
  - Kevin Pedretti
  - Mike Levenhagen
  - Kyle Wheeler
- Intel
  - Keith Underwood
  - Jerrie Coffman
  - Roy Larsen