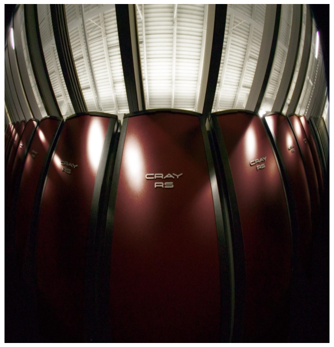
Exceptional service in the national interest



Portals as a Case Study for Holistic Comprehensive Integrated Software/Hardware Co-Design

Ron Brightwell, Technical Manager

Scalable System Software Department

Portals Interconnect Programming Interface

- 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 application protocols and libraries and system services

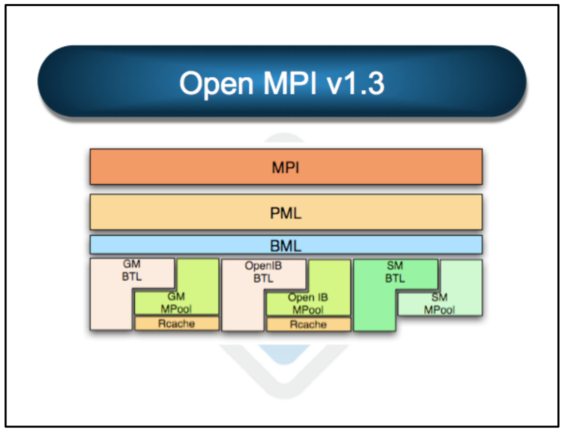
Basic Assumptions

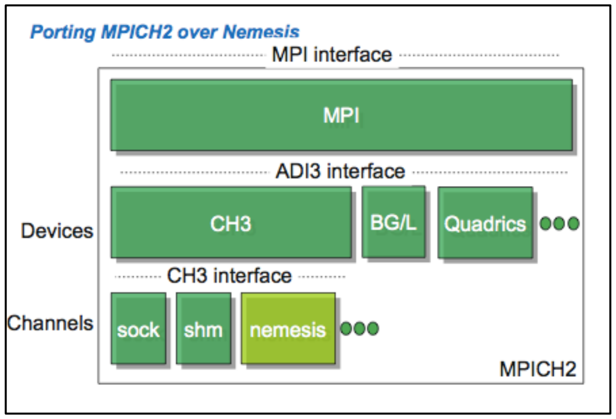
- A single low-level API is needed
  - Compute node OS doesn't have TCP/IP stack
  - Compute node application should own all network resources
- Applications will use multiple protocols simultaneously
  - Can't focus on just MPI
  - Runtime system, system call forwarding, I/O protocols too
- Need to support communication between unrelated processes
  - Client/server communication between application processes and system services
- Need to support general-purpose interconnect capabilities
  - Can't assume special collective network hardware
- Interconnect hardware limitations can't be fixed in software

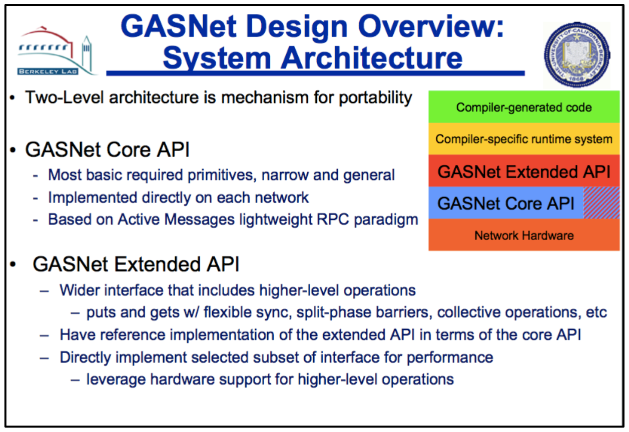
What Makes Portals Different?

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

Portals is Not Primarily a Portability Layer

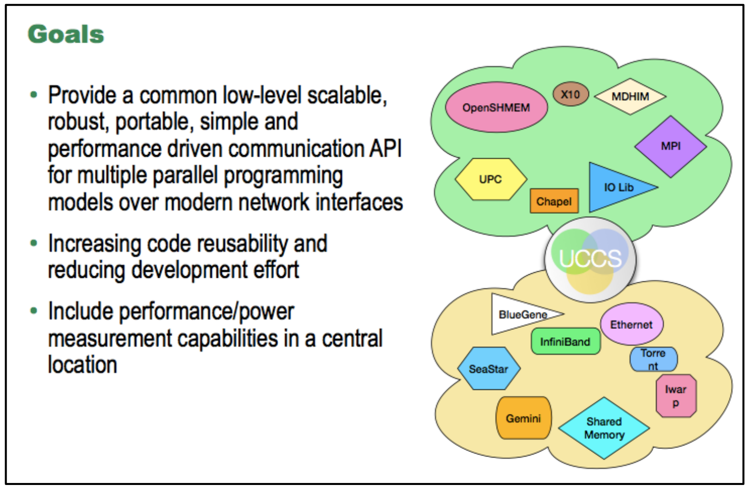





Compiler-generated code
Compiler-specific runtime system
GASNet Extended API
GASNet Core API
Network Hardware

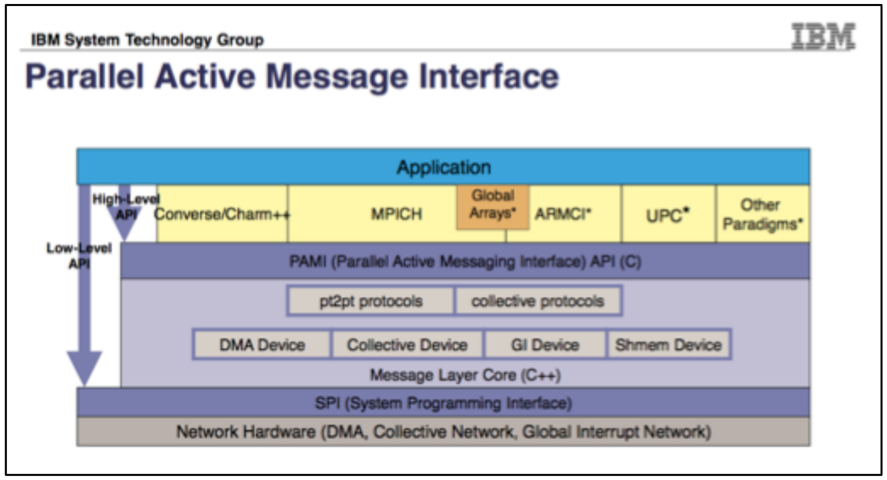
GASNet Design Overview: System Architecture

- Two-Level architecture is mechanism for portability
- GASNet Core API
  - Most basic required primitives, narrow and general
  - Implemented directly on each network
  - Based on Active Messages lightweight RPC paradigm
- GASNet Extended API
  - Wider interface that includes higher-level operations
    - puts and gets w/ flexible sync, split-phase barriers, collective operations, etc
  - Have reference implementation of the extended API in terms of the core API
  - Directly implement selected subset of interface for performance
    - leverage hardware support for higher-level operations



Goals

- Provide a common low-level scalable, robust, portable, simple and performance driven communication API for multiple parallel programming models over modern network interfaces
- Increasing code reusability and reducing development effort
- Include performance/power measurement capabilities in a central location



Parallel Active Message Interface

Design Philosophy – Don't Constrain

- Connectionless
  - Easy to do connections over connectionless
  - Impossible to do vice-versa
- One-sided
  - Easy to do two-sided over one-sided
  - Hard to do vice-versa
- Matching
  - Needed to enable flexible independent progress
  - Otherwise matching and progress must be done above
- Offload
  - Straightforward to onload API designed for offload
  - Hard to do vice-versa (see TOE)
- Progress
  - Must be implicit

Building Blocks Approach

- Define basic objects and operations that can be combined to simultaneously support multiple upper-layer protocols (ULPs)
  - Alternative approach is to define functions
  - Both approaches attempt to meet the semantics of the ULP as well as possible
- Pros
  - Supports a wider variety of upper-level protocols
  - Encapsulates important structures and functions
  - Enables specific hardware optimization opportunities
- Cons
  - More difficult to optimize for a single ULP
  - Can create interesting corner cases when combining objects and functions
  - Potential performance penalty for composability
  - Exposes implementation details

ULPs Supported

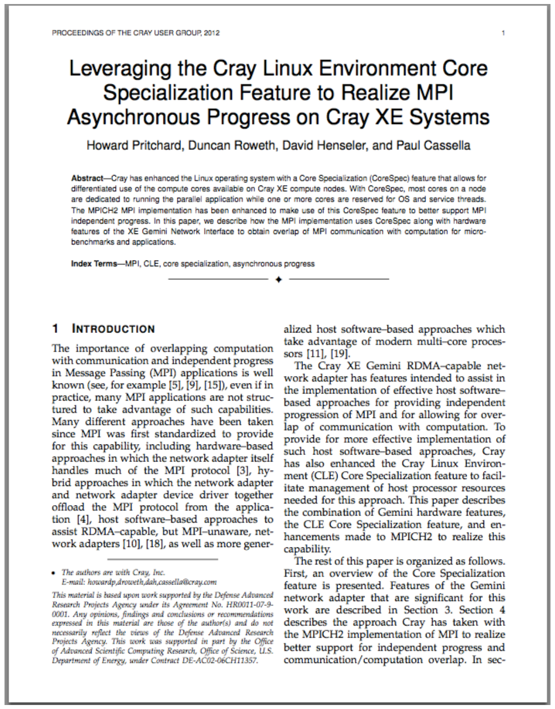
- Application services
  - MPI-1, MPI-2, MPI-3 (send/recv, collective, one-sided)
    - MPICH, MPI/Pro, ChaMPIon/Pro, MPICH2, OpenMPI
  - PGAS
    - Cray SHMEM, OpenSHMEM, GASNet, ARMCI
  - MultiProcessor Computing (MPC)
  - CCI
- OS/Runtime services
  - Parallel job launch
    - Yod
  - File system and I/O
    - Fyod, Lustre
  - System calls
    - Remote procedure calls
  - IP
  - Qthreads runtime

Onload Versus Offload

- Why design a custom NIC for offload?
- Just dedicate a core
  - A 3 GHz Xeon will outperform a 500 MHz embedded processor on network protocol processing
  - A custom ASIC is even more expensive
- Cost will go down as core count increases

Cray Core Specialization

- Dedicate "OS" cores to handle MPI progress
  - MPI progress threads run on a dedicated set of cores



PROCEEDINGS OF THE CRAY USER GROUP, 2012 1

Leveraging the Cray Linux Environment Core Specialization Feature to Realize MPI Asynchronous Progress on Cray XE Systems

Howard Pritchard, Duncan Roweth, David Henseler, and Paul Cassella

Abstract—Cray has enhanced the Linux operating system with a Core Specialization (CoreSpec) feature that allows for differentiated use of the compute cores available on Cray XE compute nodes. With CoreSpec, most cores on a node are dedicated to running the parallel application while one or more cores are reserved for OS and service threads. The MPICH2 MPI implementation has been enhanced to make use of this CoreSpec feature to better support MPI independent progress. In this paper, we describe how the MPI implementation uses CoreSpec along with hardware features of the XE Gemini Network Interface to obtain overlap of MPI communication with computation for micro-benchmarks and applications.

Index Terms—MPI, CLE, core specialization, asynchronous progress

1 INTRODUCTION

The importance of overlapping computation with communication and independent progress in Message Passing (MPI) applications is well known (see, for example [5], [9], [15]), even if in practice, many MPI applications are not structured to take advantage of such capabilities. Many different approaches have been taken since MPI was first standardized to provide for this capability, including hardware–based approaches in which the network adapter itself handles much of the MPI protocol [3], hybrid approaches in which the network adapter and network adapter device driver together offload the MPI protocol from the application [4], host software–based approaches to assist RDMA–capable, but MPI–unaware, network adapters [10], [18], as well as more generalized host software–based approaches which take advantage of modern multi–core processors [11], [19].

The Cray XE Gemini RDMA–capable network adapter has features intended to assist in the implementation of effective host software–based approaches for providing independent progression of MPI and for allowing for overlap of communication with computation. To provide for more effective implementation of such host software–based approaches, Cray has also enhanced the Cray Linux Environment (CLE) Core Specialization feature to facilitate management of host processor resources needed for this approach. This paper describes the combination of Gemini hardware features, the CLE Core Specialization feature, and enhancements made to MPICH2 to realize this capability.

The rest of this paper is organized as follows. First, an overview of the Core Specialization feature is presented. Features of the Gemini network adapter that are significant for this work are described in Section 3. Section 4 describes the approach Cray has taken with the MPICH2 implementation of MPI to realize better support for independent progress and communication/computation overlap. In sec-

- *The authors are with Cray, Inc.*
  E-mail: howardp,droweth,dah,cassella@cray.com

This material is based upon work supported by the Defense Advanced Research Projects Agency under its Agreement No. HR0011-07-9-0001. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Defense Advanced Research Projects Agency. This work was supported in part by the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract DE-AC02-06CH11357.

S3D Time Step Summary

# Application Threads	Progression disabled	Progression enabled
14	4.77	3.93
15	4.68	4.05
16	4.59	4.06

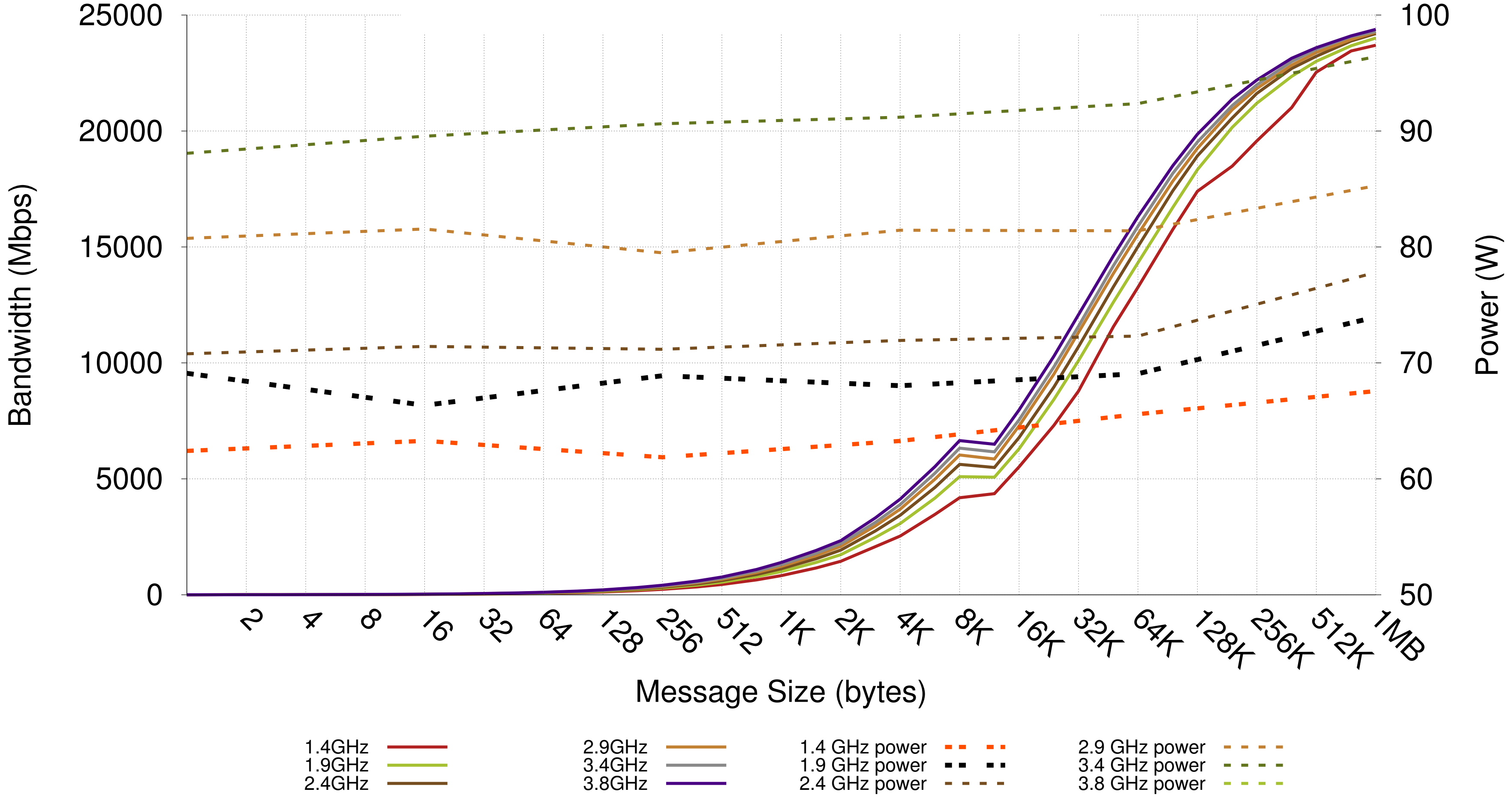
MILC Run Time Summary(secs)

# Run Type	4096 ranks	8192 ranks
No progression	2165	1168
Progression (phase 1)	2121	1072
Progression (phase 2)	3782	2138
Progression (phase 1) no reserved cores	3560	2210
Progression (phase 1) reserve core but no corespec	2930	2070

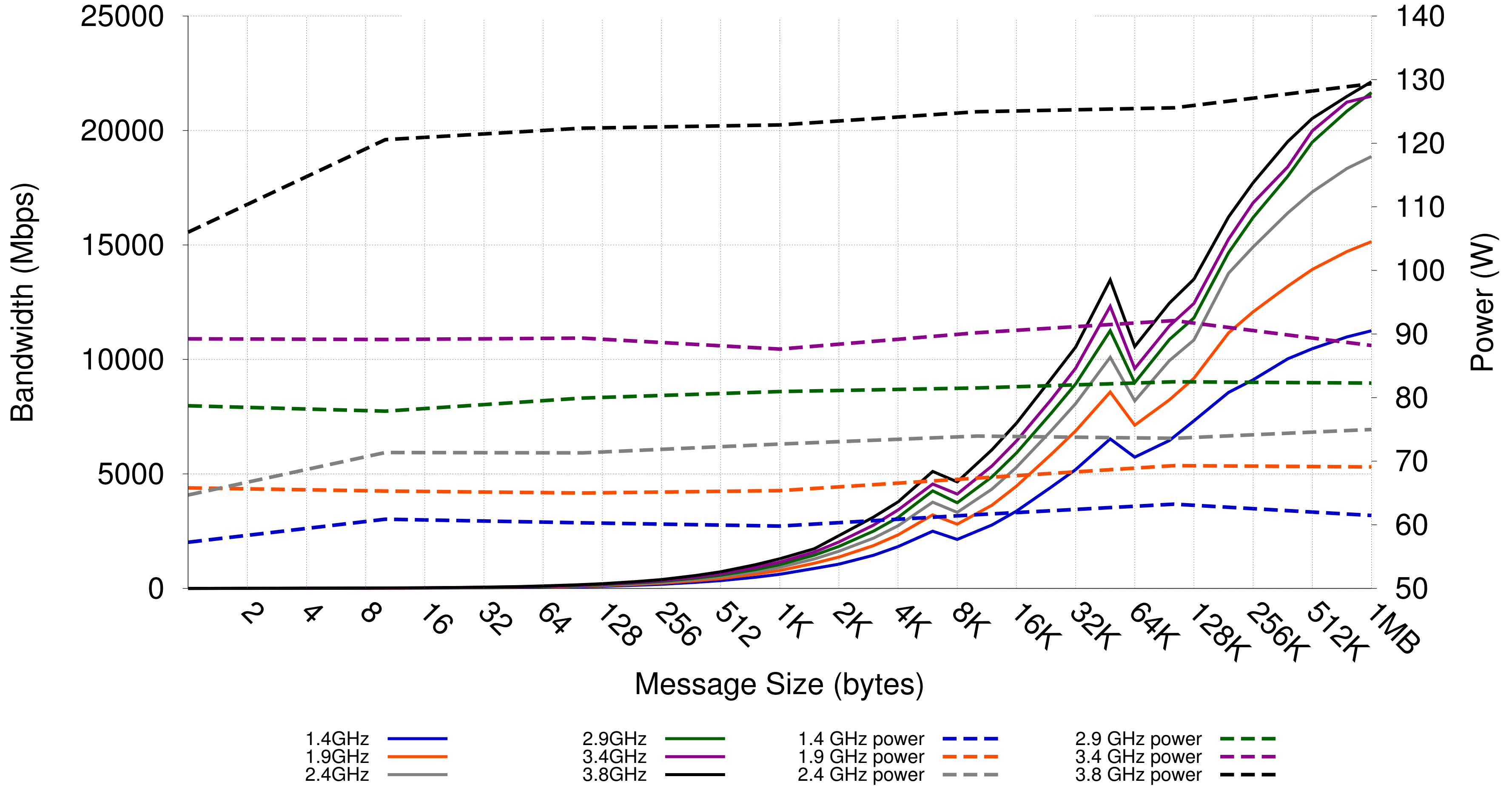
Onload Versus Offload Evaluation

- Tests conducted on the Sandia Teller testbed
  - 3.8 GHz AMD Piledriver quad-core processors
- PowerInsight measurement boards
- Onload NIC
  - Qlogic InfiniBand HCAs
- Offload NIC
  - Mellanox InfiniBand HCAs
- Performance results using Netpipe-3.7.1

Offload Streaming Bandwidth Results



Onload Streaming Bandwidth Results



Portals 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

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 approach
  - 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

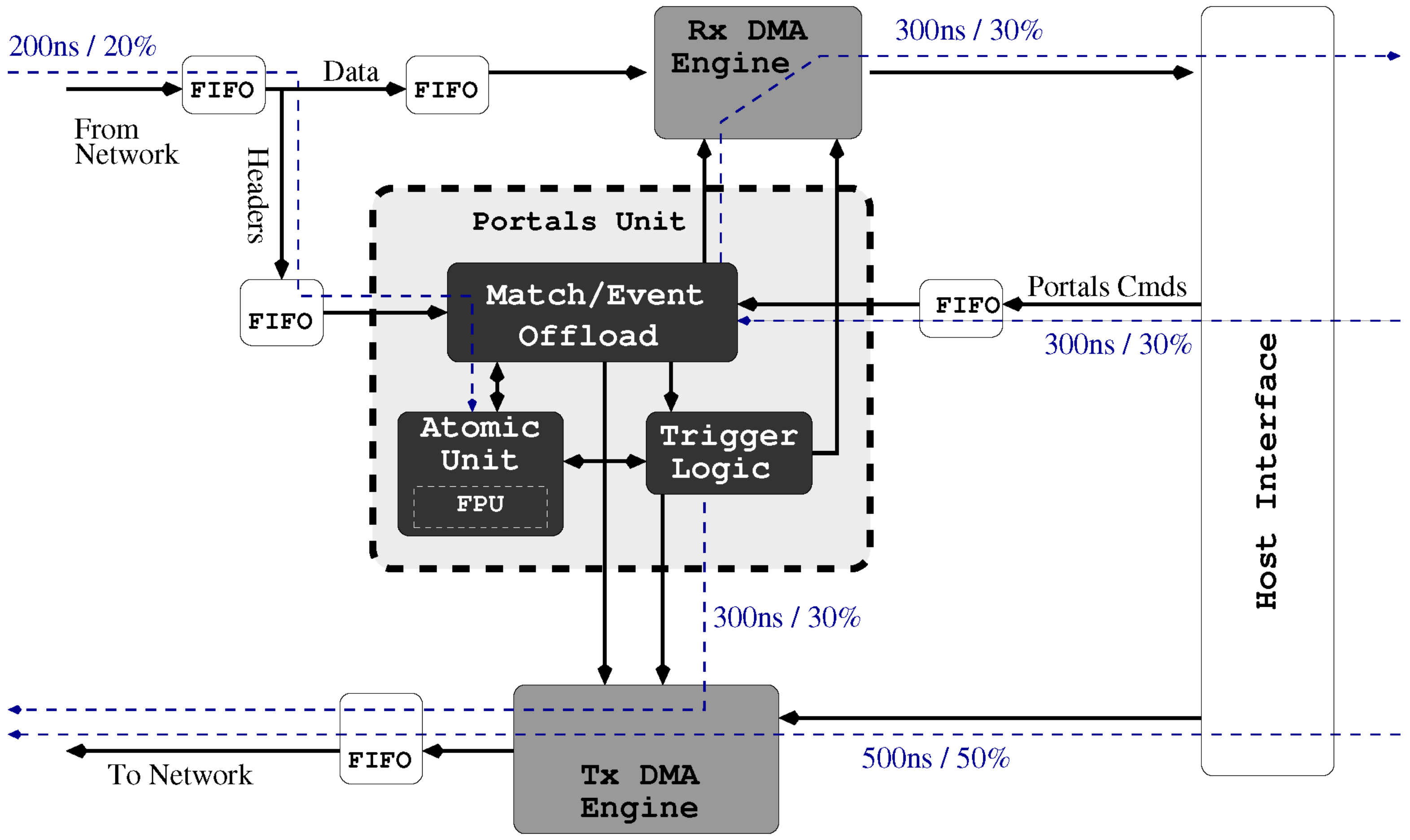
Generality of Triggered Operations

- 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

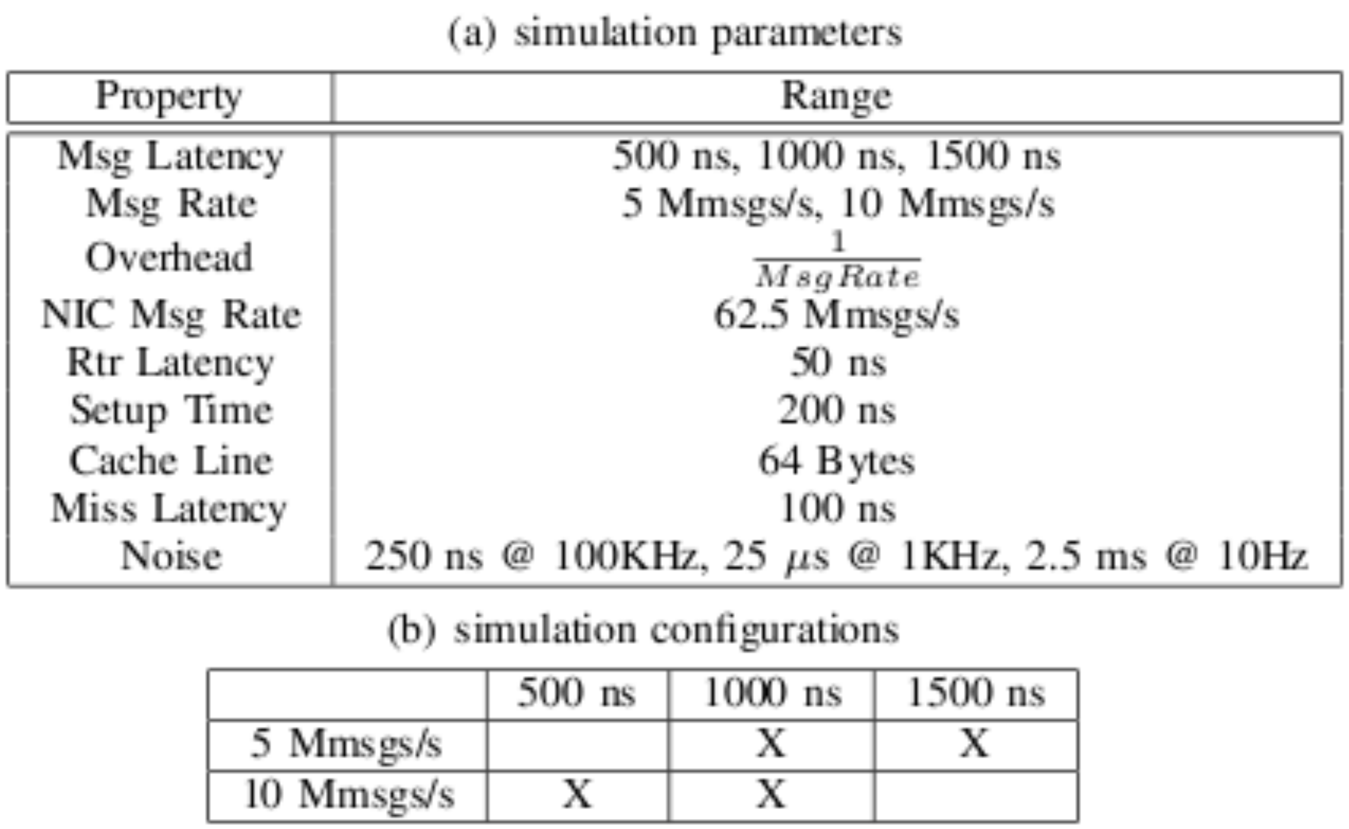
Simulation Methodology

- 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



Simulation Settings



(a) simulation parameters

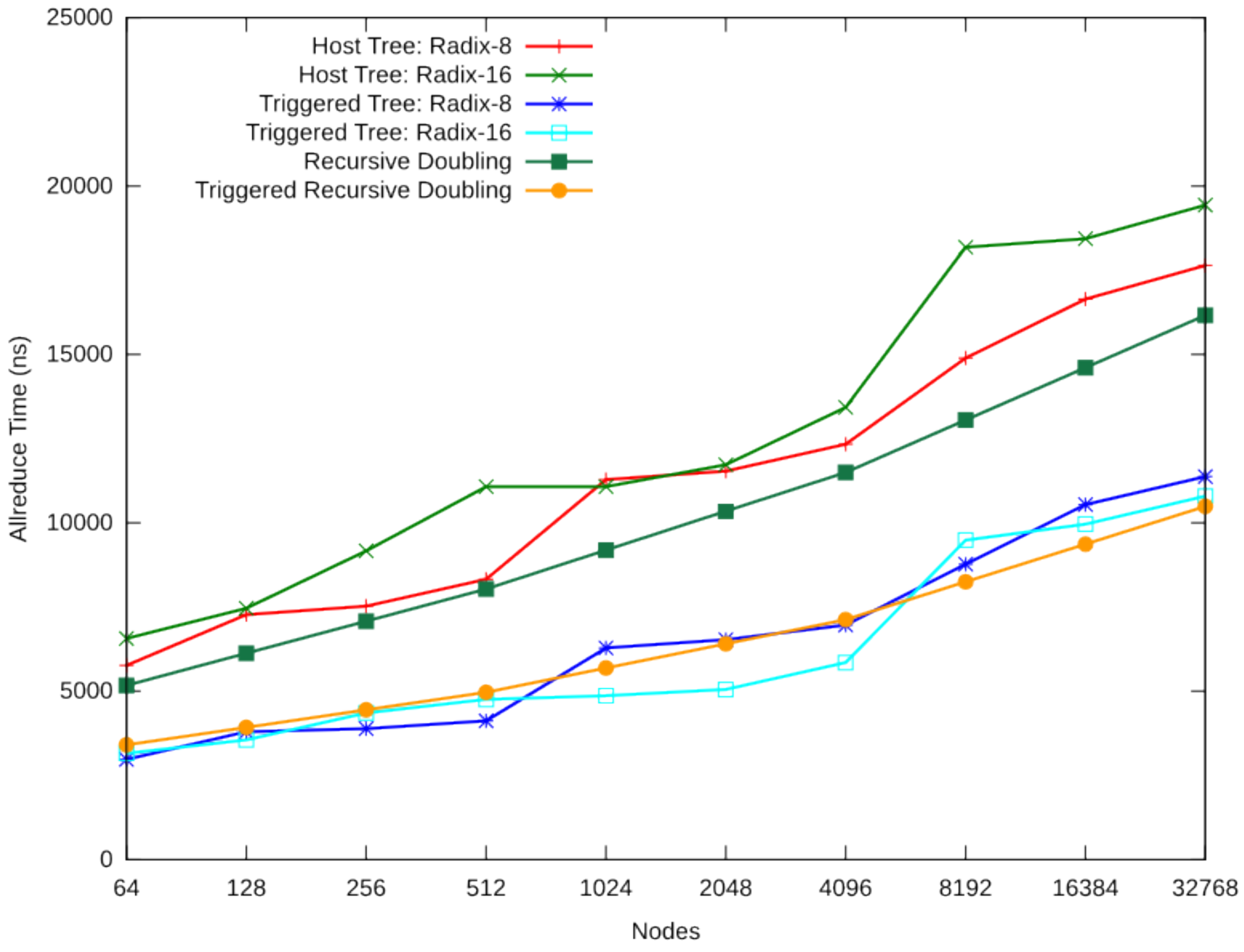
Property	Range
Msg Latency	500 ns, 1000 ns, 1500 ns
Msg Rate	5 Mmsgs/s, 10 Mmsgs/s
Overhead	$\frac{1}{MsgRate}$
NIC Msg Rate	62.5 Mmsgs/s
Rtr Latency	50 ns
Setup Time	200 ns
Cache Line	64 Bytes
Miss Latency	100 ns
Noise	250 ns @ 100KHz, 25 μs @ 1KHz, 2.5 ms @ 10Hz

(b) simulation configurations

	500 ns	1000 ns	1500 ns
5 Mmsgs/s		X	X
10 Mmsgs/s	X	X	

Allreduce

500ns, 10 Mmsgs/s

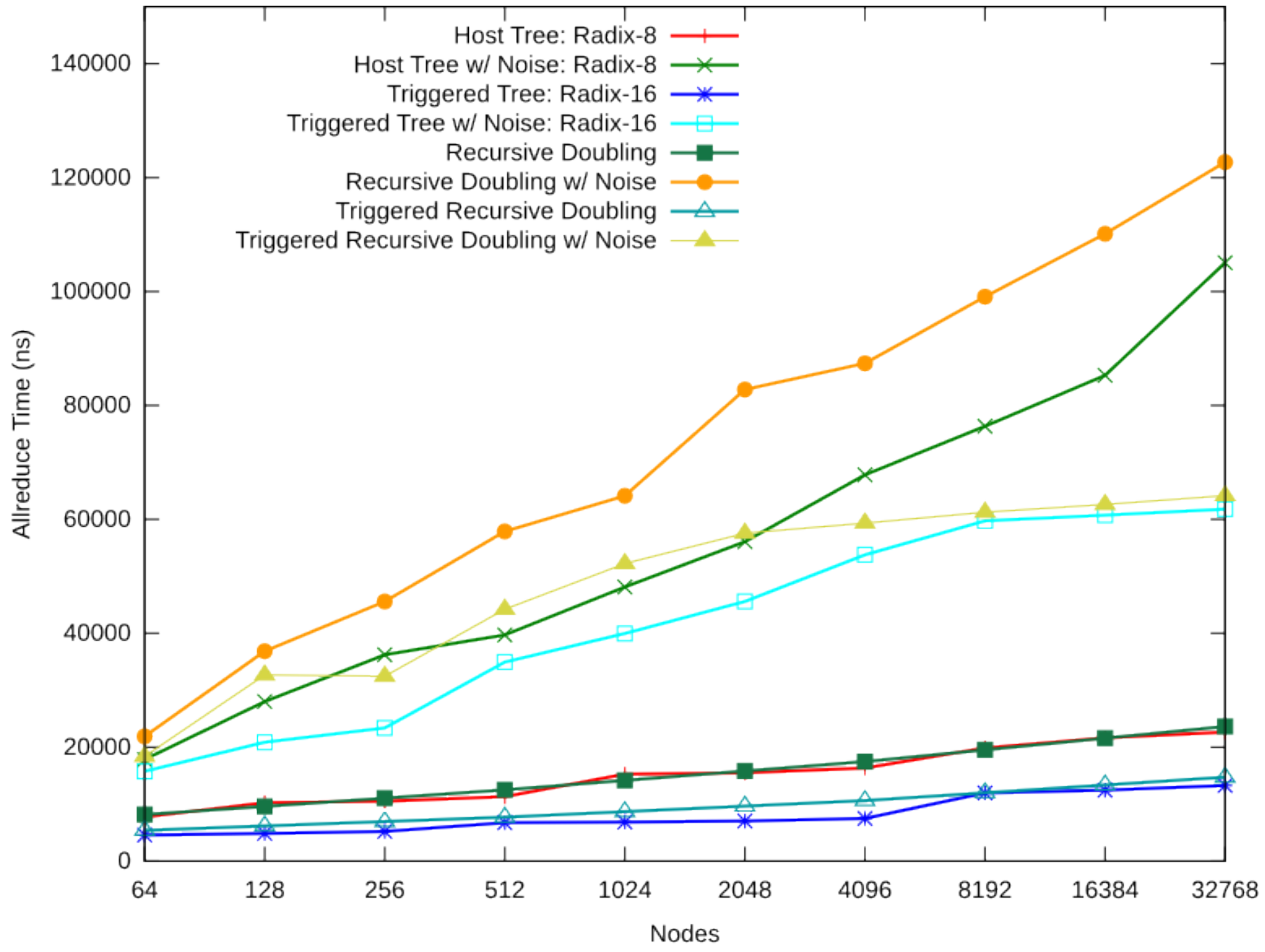


Noise Simulations

- 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

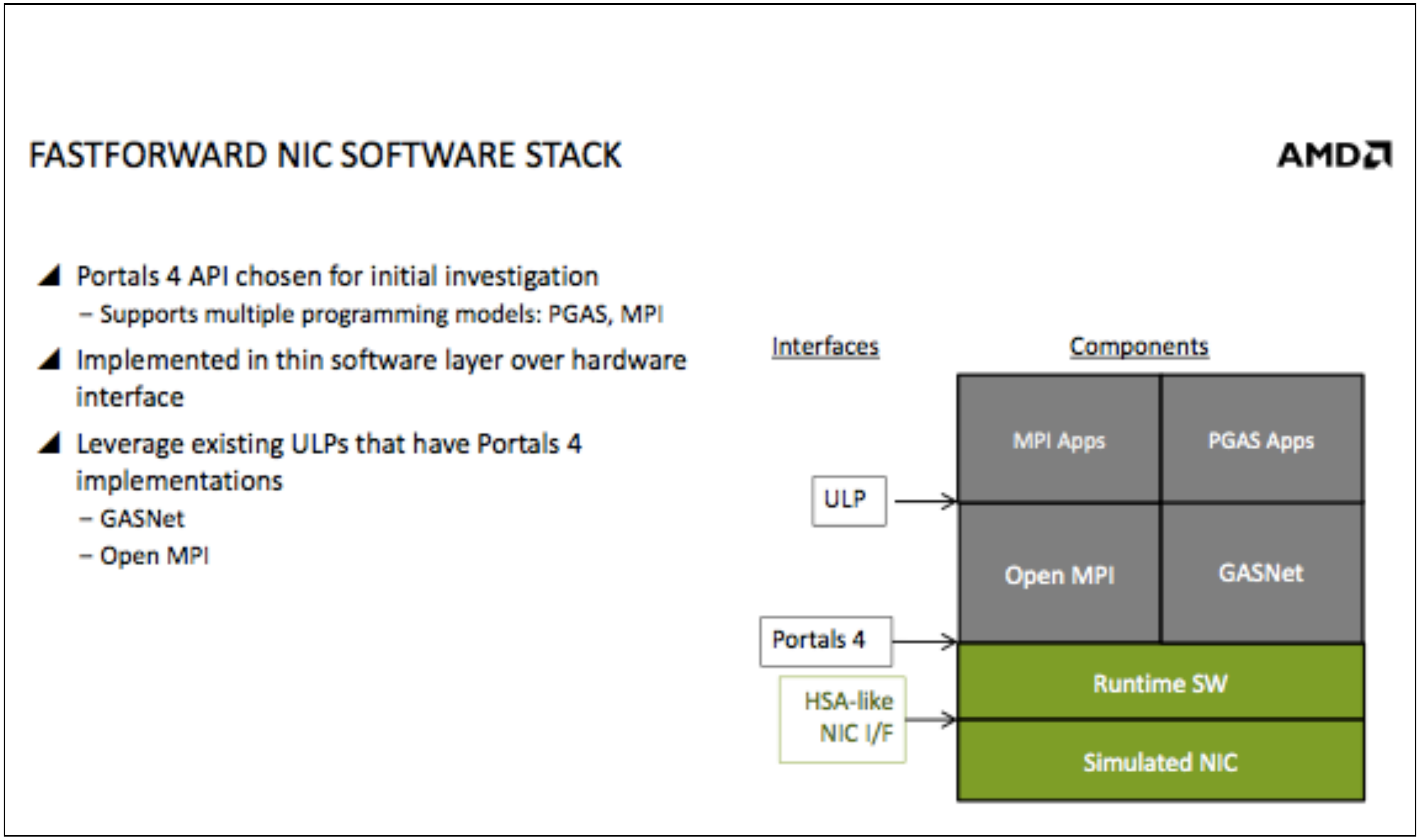
Interesting Things We Learned

- 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

AMD Using Portals 4 for Co-Design



FASTFORWARD NIC SOFTWARE STACK

AMD

- Portals 4 API chosen for initial investigation
  - Supports multiple programming models: PGAS, MPI
- Implemented in thin software layer over hardware interface
- Leverage existing ULPs that have Portals 4 implementations
  - GASNet
  - Open MPI

Interfaces
Components
MPI Apps
PGAS Apps
ULP
Open MPI
GASNet
Portals 4
Runtime SW
HSA-like NIC I/F
Simulated NIC

Summary

- Portals 4 provides building blocks that support many ULPs
- Encapsulates required semantics in a single API
- Design decisions based on least constraints
- Reference implementation available
  - Trying to figure out how to not be just another layer of software
  - Reference implementation performance should always be bad
- Triggered operations can implement
  - Non-blocking collective operations
  - Efficient rendezvous protocol for long messages
  - Recovery-based flow control for MPI
- Simple cores may not support network onload very well

Acknowledgments

- Sandia
  - Ryan Grant
  - Scott Hemmert
  - Kevin Pedretti
  - Mike Levenhagen
- Intel
  - Keith Underwood
  - Jerrie Coffman
  - Roy Larsen
- Amazon
  - Brian Barrett
- Micron
  - Kyle Wheeler

http://www.cs.sandia.gov/Portals