SAND2016-4296PE

Exceptional service in the national interest

Sandia National Laboratories

Extreme Computing

Pushing the Frontiers of Science

U.S. DEPARTMENT OF ENERGY

NNSA National Nuclear Security Administration

Images courtesy of: Oak Ridge National Laboratory; Argonne National Laboratory and Cray Inc.
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. SAND NO. 2016-

Outline

- HPC Overview
- Challenges facing HPC for Exascale
  - Scalability (networks)
  - Power
- Current HPC environment
- High Performance Networks Deep Dive
  - Where we are now
  - Research (UNM collaborations)
- Power Challenges
  - Research (SNL, SNL/UNM collaborations)
- Conclusions
- Questions

High Performance Computing

- What is it?
  - Massively parallel systems, typically made of many small commodity servers networked together, performing a few large or many small tasks.
- Why do we need it?
  - You use supercomputers everyday:
    - Google (capacity computing)
    - Weather forecasts (capability)
    - Amazon (capacity computing)
  - And the products of research conducted with supercomputers
    - Oil discovery and extraction
    - Drug research
    - Fundamental science (physics, chemistry, etc.)

Exascale: Why?

- Why do we need an Exascale computer?

- Explore new areas of science
  - Whole Earth climate modeling
    - Better understand the effects of climate change
  - Whole Earth system modeling
    - Energy and Economic modeling
  - New high-energy/Quantum physics questions
    - Huge datasets from CERN
  - Simulating the human brain
    - Currently we can simulate a rat's brain, or a simplified cat brain

Moore's Law – Current Challenges

- Growing transistor density has continued despite a halt in increasing clock frequency
  - Vendors provide more CPU cores on the same chip
  - How do we use these cores effectively?

- Alternative architectures like GPUs and many core processors like KNL (Intel Phi)
  - Can we take advantage of many-core architectures?
  - What bottlenecks exist?
  - What applications map well to alternative computational architectures?

Moore’s Law – Future Challenges

- Fetching data for the cores becomes the dominant activity in terms of energy.
- Memory per core is dropping
- Memory becomes a major performance and energy bottleneck
- Some alternatives have been proposed
  - Hybrid Memory Cube consortium
    - Vertically integrated DRAM over a logic base with silicon vias
    - Much higher bandwidth (~240GB/s)
    - Lower power consumption (70-90%)
  - High Bandwidth Memory (HBM)
    - Bandwidth (up to 256 GB/s)
  - Both are TSV reliant

Exascale Challenges

- Exascale provides a unique challenge
- Many existing models will not work well for Exascale
- One of the largest challenges is meeting the power budget (20-30 MW)
- We could build an Exascale like system out of current technology, but it would require 100s of MWs of power

History of Supercomputing

- Exascale computing is hitting a power/reliability wall
- We hit a power/reliability wall in the 1950s with vacuum tubes
  - Many thought ENIAC would never work due to the unreliability of vacuum tubes
- Solved the power-reliability problem with the transistor
  - Power/heat problem is back again!
- Miniaturization made possible by integrated circuits and printed circuit boards

Exascale Challenges

- Networks: The focus of this talk!
  - How do that many threads communicate effectively?
  - Are we still going to use a message passing + X type programming model for communication?
    - It looks like we're stuck with MPI
    - Too much money/time has been invested in MPI programs
  - How does the message passing model effectively scale to Exascale sizes?
    - One leading interconnect, InfiniBand, has issues with scalability
    - Can we even match messages in software anymore?

Exascale Challenges

- Applications:
  - How can applications take advantage of billion way parallelism?
    - What subset of applications are even viable at Exascale?
  - How do we debug a billion thread program?
    - We need some automated tool assistance
    - Current parallel debuggers are not up to the task
    - New debuggers must be fast enough to be tolerable
      - Cannot have multi-minute latency on a single debugger step/command

Exascale Computing

- Key challenge approaches from SNL:
  - Operating System
    - Kitten – Lightweight OS for supercomputing
      - Successor to the long line of lightweight HPC kernels (SUNMOS, puma, cougar, catamount)
  - Networks
    - Portals
      - Bull BXI – Portals hardware ASIC
      - Provides matching in hardware (as opposed to IB)
      - Low latency, high bandwidth
      - Designed to support MPI
  - Threads
    - Qthreads
      - Many lightweight threads overseen by “shepherds”
      - Scheduling approaches for HPC, including power/energy concerns

Exascale Computing

- The power budget: not just about energy
  - The relative cost of energy is important, but not the #1 concern
  - Peak power is a major issue
    - Constrained to 20-30 MW? How much can we fluctuate up/down?
    - Issues for power generation facilities for large consumption swings over very small time periods
- How do we cool off this 20 MW space heater?
  - Traditional air cooling
  - Cold water cooling
  - Warm/hot water cooling (use the waste heat for heating buildings)
  - Exotic cooling technologies (direct Freon, liquid gases)

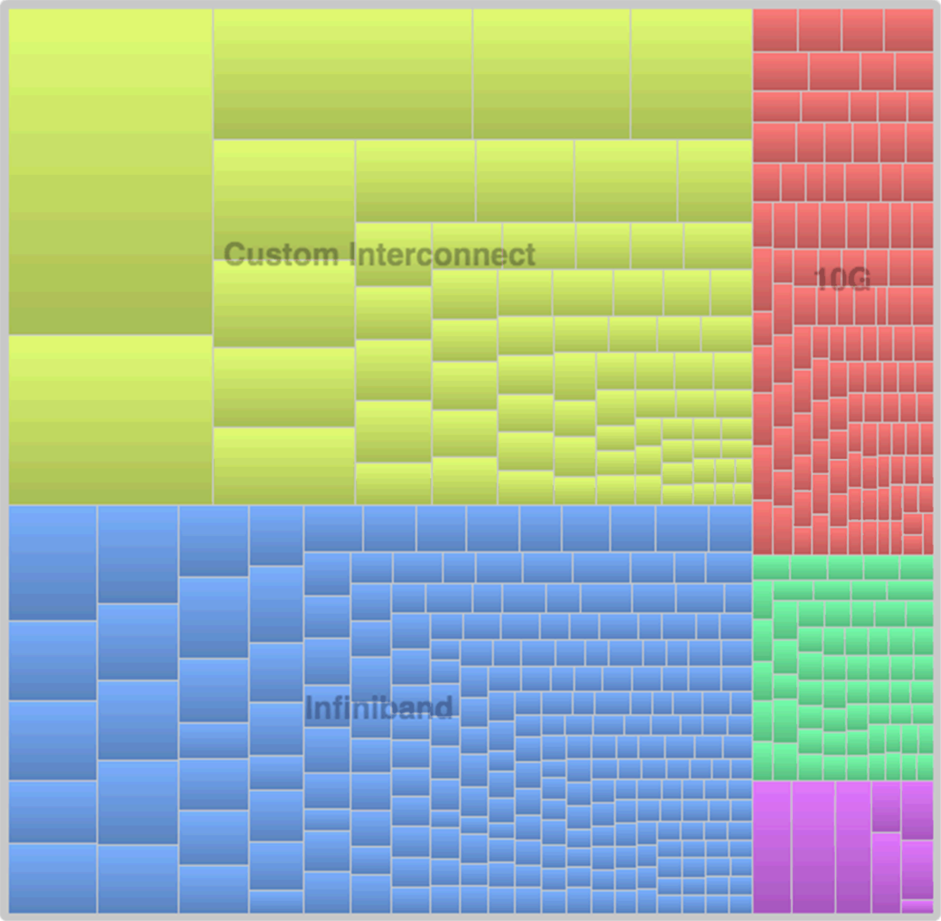
Exascale Computing

- Due to reliance on commodity components:
  - Industry requires direction now
  - Investment in alternative technologies (ex. Hybrid memory cubes)
  - Decisions will affect consumer products over the next 10 years

- The commodity market drives the HPC market
  - Cloud computing is bringing commercial computing closer in line with traditional HPC
  - Cloud apps are still capacity-apps, making use of parallelism easier than for capability HPC applications.

Exascale Computing

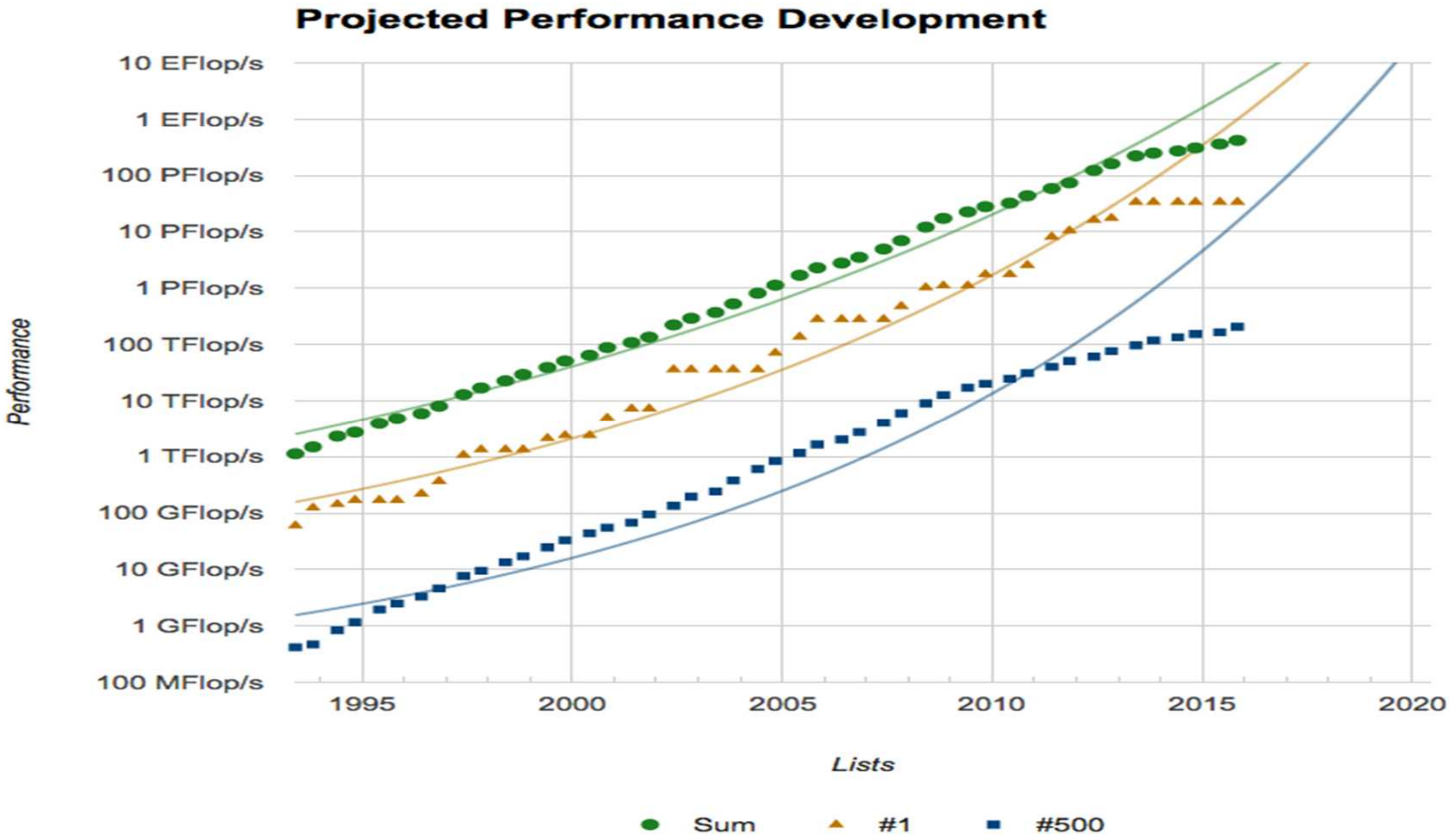
- Networks: Where we are now



Source:
www.top500.org

Exascale Computing

- Processing: Where we are now

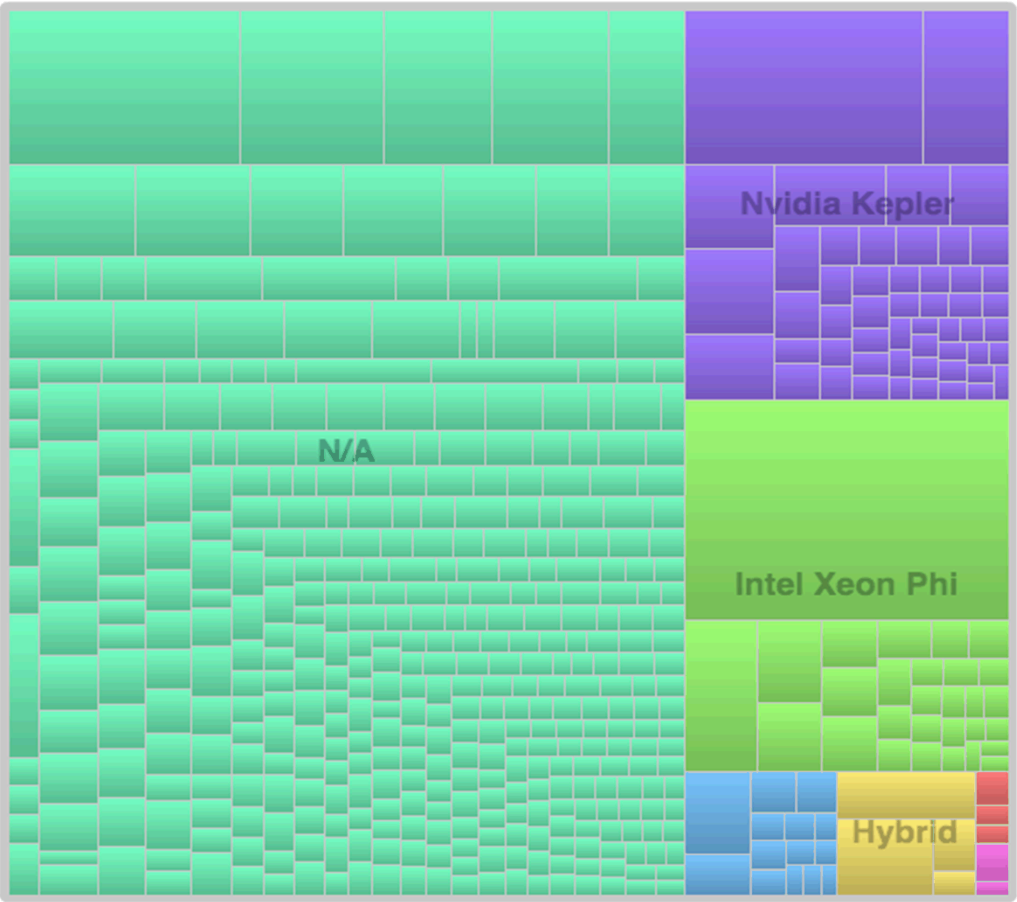


Projections Graph courtesy: www.top500.org

Exascale Computing

- Processing: where we are now

Trending towards accelerators/alternative architectures



Source: www.top500.org

Exascale Computing

- Many-core is interesting, yet unknown

- Benefits:
  - Massive parallelism,
  - Low power consumption,
  - Easy communication between local Accelerator/CP cores

- Drawbacks:
  - Massive parallelism at the expense of core complexity
  - Getting data to/from the cores can be difficult/slow/expensive (KNC)
  - New programming paradigms (CUDA, OpenCL, Intel Phi)

Exascale Computing

- Many-core be a method to reach Exascale
  - Intel Phi is essentially a bunch of heavily modified Pentium CPUs
  - Added vector support
  - Extra cache size/levels
  - No legacy SIMD support (MMX etc.)
  - Good scalability, individual core performance could be an issue
    - 1 Teraflop in a single KNC card (300W TDP)
  - First teraflop supercomputer was ASCI Red (SNL) in June 1997, remained the fastest computer until late 2000.
    - 9298 Pentium II processors, 1600 sqft, 850 kW of power.

Exascale Computing

Extreme Scale Communications at SNL

Brief Introduction to Portals

- Portals – Scalable Networking API developed at Sandia
  - More than 20 years in development
  - Each iteration = major changes

- Portals 1, 2, and 3 used in large systems
  - Portals 3 network for most commercially successful Cray machine
  - Seastar interconnect was used for many years

- Portals is designed for MPI
  - Also support for one-sided SHMEM-type models

- Portals 4
  - Native Network API for Bull’s BXI interconnect

Portals 4 – Modern Network API

- Portals has evolved since Seastar
  - Better support for one-sided communications
  - Tweaks to API to prevent protocol quirks that lead to slowdown
- Portals 4 has rich support for
  - Triggered operations (key for complex offloading)
  - Atomic operations
  - One-sided communications
- Portals 4.0.2 released last year
  - 4.1 coming this calendar year
  - 4.2 close after that depending on HW vendor capabilities

Current HPN APIs

- Verbs – current interface for InfiniBand and iWARP devices
  - Developed originally as an API for high performance storage
  - Has technical/semantic mismatches with HPN needs
  - Very verbose programming API
- UGNI – current interface for Cray Gemini/Aires networks
  - Specifically for Gemini/Aires hardware
  - Short message optimizations (GNI Short message SMSG facility)
- Portals4 – open interface specification from SNL, interface for Bull BXI
  - Scalable connectionless API
  - Unexpected message support
  - Message matching operations
  - Asynchronous (independent) progress

Why not Verbs?

- Simple communication code (send from one peer to another)
  - Verbs: > 650 lines of code
  - Portals: ~125 lines of code

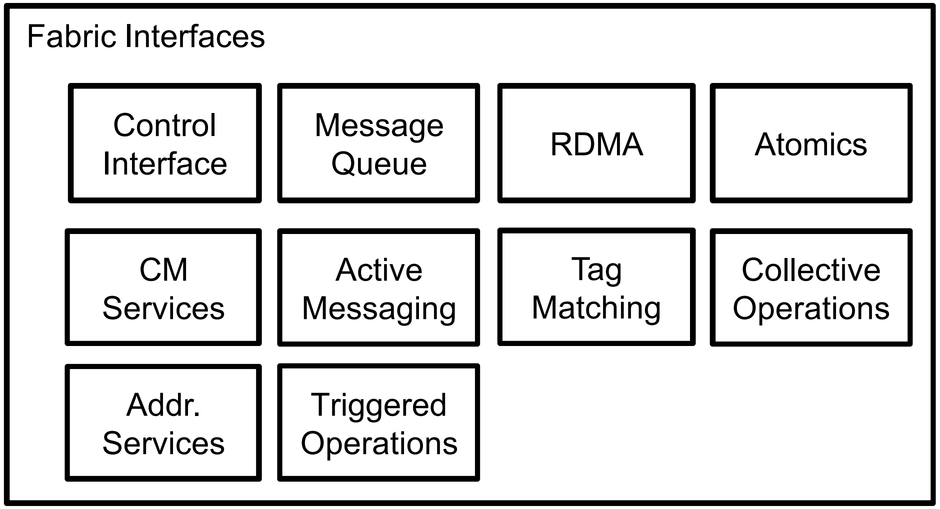
- Does not support features like Tag Matching or Active Messages

- Memory registration nightmare
  - Forced to manage/pin currently registered memory

- Very hardware centric API
  - Can cause application development difficulties

OFI

- Open Fabrics Interface (not Verbs 2.0)
  - Sockets-like API for interfacing with multiple “providers”
  - Providers can support a subset of possible interfaces
  - Libfabric will continue to support Verbs, just hide some of the details from the application
  - Automatic and implicit progress models



Fabric Interfaces

Control Interface

Message Queue

RDMA

Atomics

CM Services

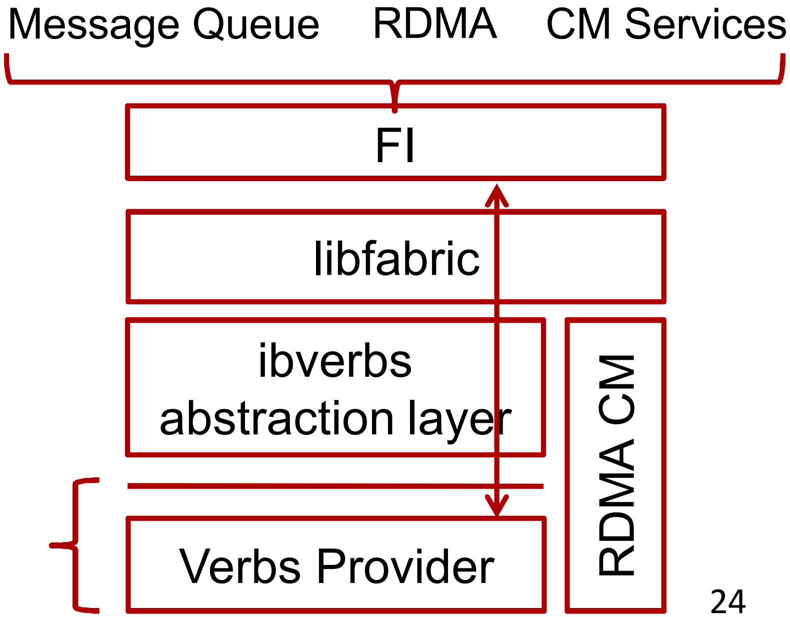
Active Messaging

Tag Matching

Collective Operations

Addr. Services

Triggered Operations



Message Queue

RDMA

CM Services

FI

libfabric

ibverbs abstraction layer

Verbs Provider

RDMA CM

Figures source: OFIWG archives, Open Fabrics 2.0 Sean Hefty, Intel Corp. http://downloads.openfabrics.org/downloads/ofiwg/ofi_architecture/ofwg-fi.pptx

Verbs vs. OFI vs. Portals

	Verbs	OFI	Portals
Code Verbosity	Poor	Good	Good
Semantics Match	Poor	Good	Excellent
OS bypass	Yes	Yes	Yes
Application bypass	No	Possible (if supported)	Yes
Connection model	Connections & Unreliable Connectionless	Connections/Connectionless	Connectionless
Tag Matching	No	Yes (if supported)	Yes
Hardware Interface	Yes	No (needs providers)	Yes
Hardware Collectives	Yes (core direct)	Yes (if supported)	No

Understanding HPN

- Designing/Developing network APIs requires research
  - Understand the implications of design choices
- Need to understand network design tradeoffs with new core architectures and communication mechanisms

Work with UNM on network research

- Do we need sophisticated NIC architectures?
- How does parallelism in the communication stack impact performance?
- How well do alternative data movement techniques work?
- What are possible ramifications to alternative data movement strategies?

Exascale Computing

Extreme Scale Communications Research

Do we need sophisticated NICs?

- Understanding Hardware vs. Software approaches

- Onload (software) vs. Offload (hardware)

- Tradeoffs need to be understood for architectural changes
  - Impact of many-core on networking

- Motivate/demotivate use of offloading approaches like Portals

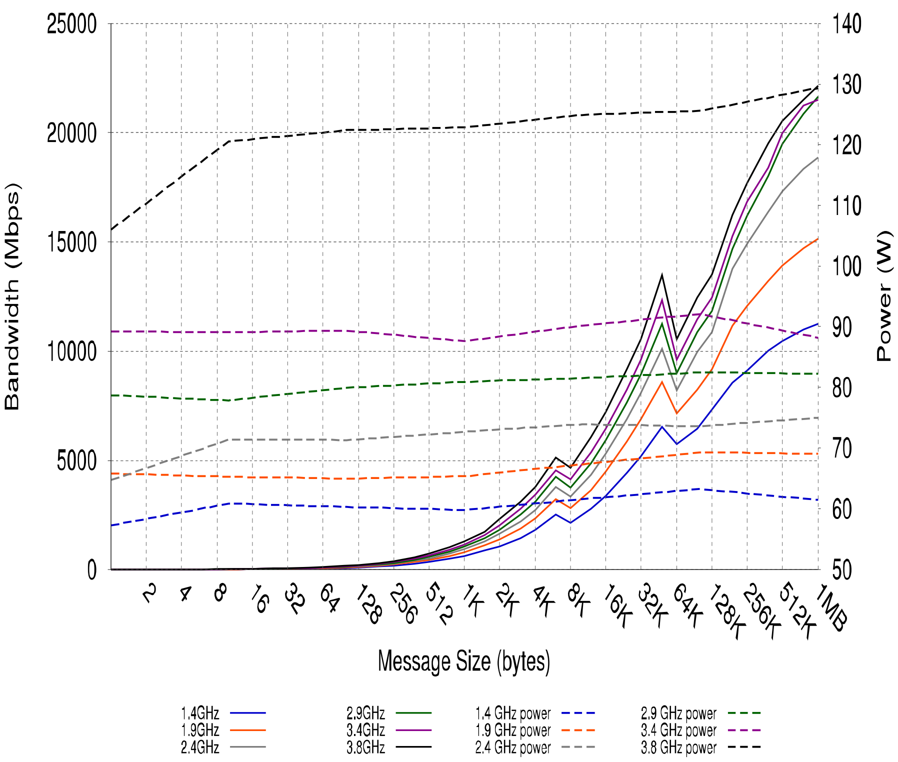
Matthew Dosanjh, Ryan E. Grant, Patrick Bridges, Ron Brightwell, “Re-evaluating Network Onload vs. Offload for the Many-Core Era”, IEEE International Conference on Cluster Computing (Cluster 2015), Chicago, IL, USA, September 8-11, 2015. (Acceptance Rate: 24%).

Offload vs. Onload

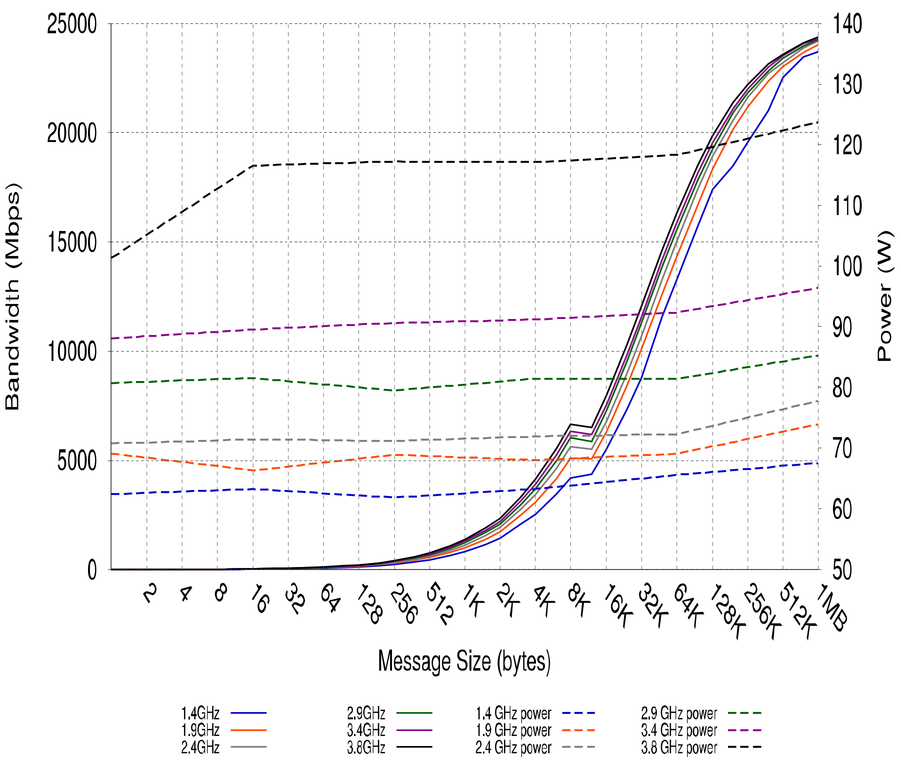
- Onloading allows for very easy matching between the hardware and software layer semantics
- Offloading can provide better performance if well matched to the application semantics without consuming computational resources on the host
- Onloading relies on an abundance of (dedicated) high capability fast cores to provide good network performance
- Offloading can provide hardware optimized approaches to known operations (e.g. content addressable memory lookups for matching)

Onload vs. Offload

- Decreasing core frequency also impacts message rate
  - Depends on how much HW support you have to help you (1.5% vs. 35.1% performance loss, offload vs. onload)



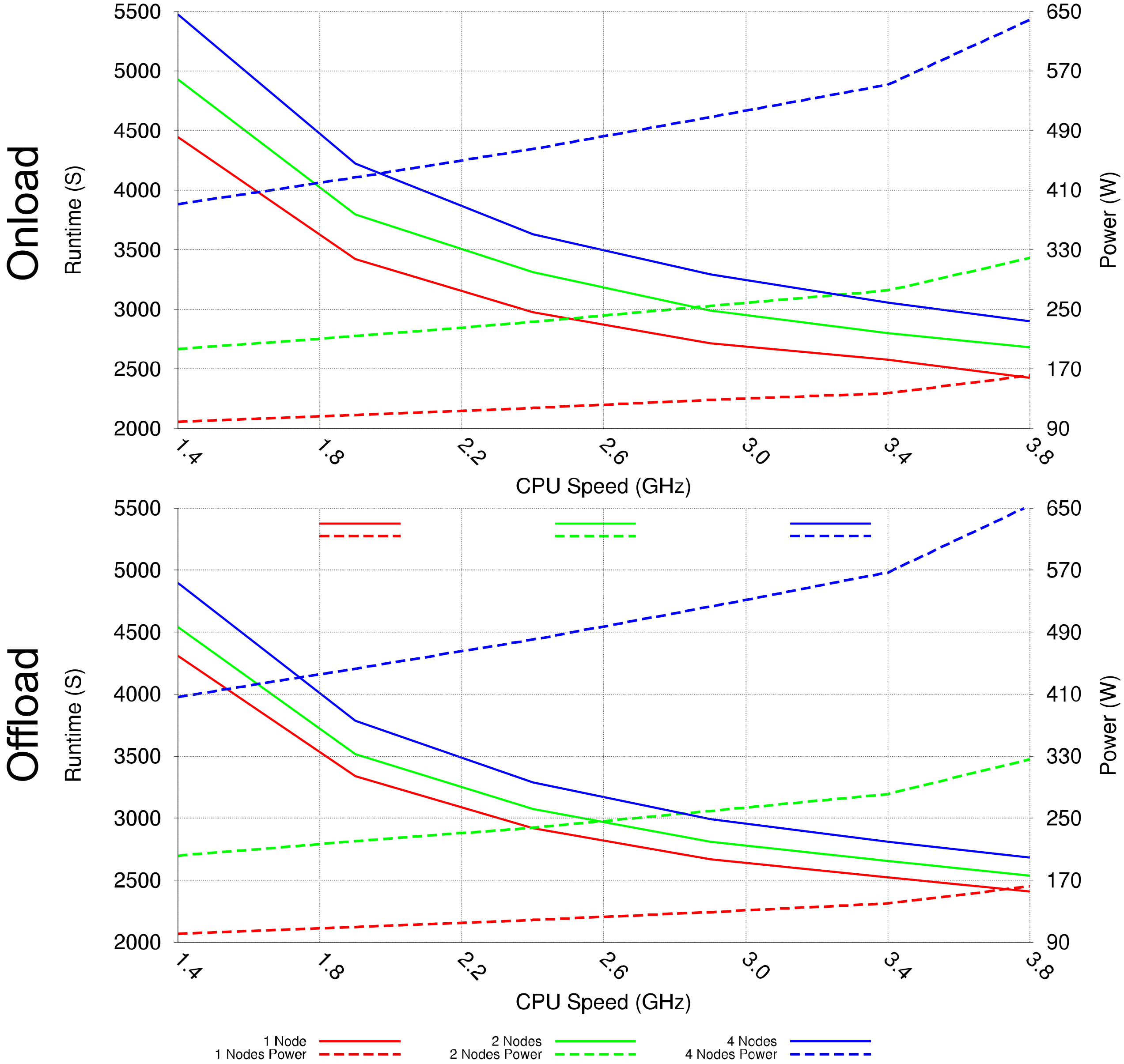
Onload Stream Bandwidth (Put) With Power



Offload Stream Bandwidth (Put) With Power

M. G. F. Dosanjh, R. E. Grant, P. G. Bridges, and R. Brightwell. Reevaluating network onload vs. offload for the many-core era. In IEEE Intl. Conf. on Cluster Computing. IEEE, 2015

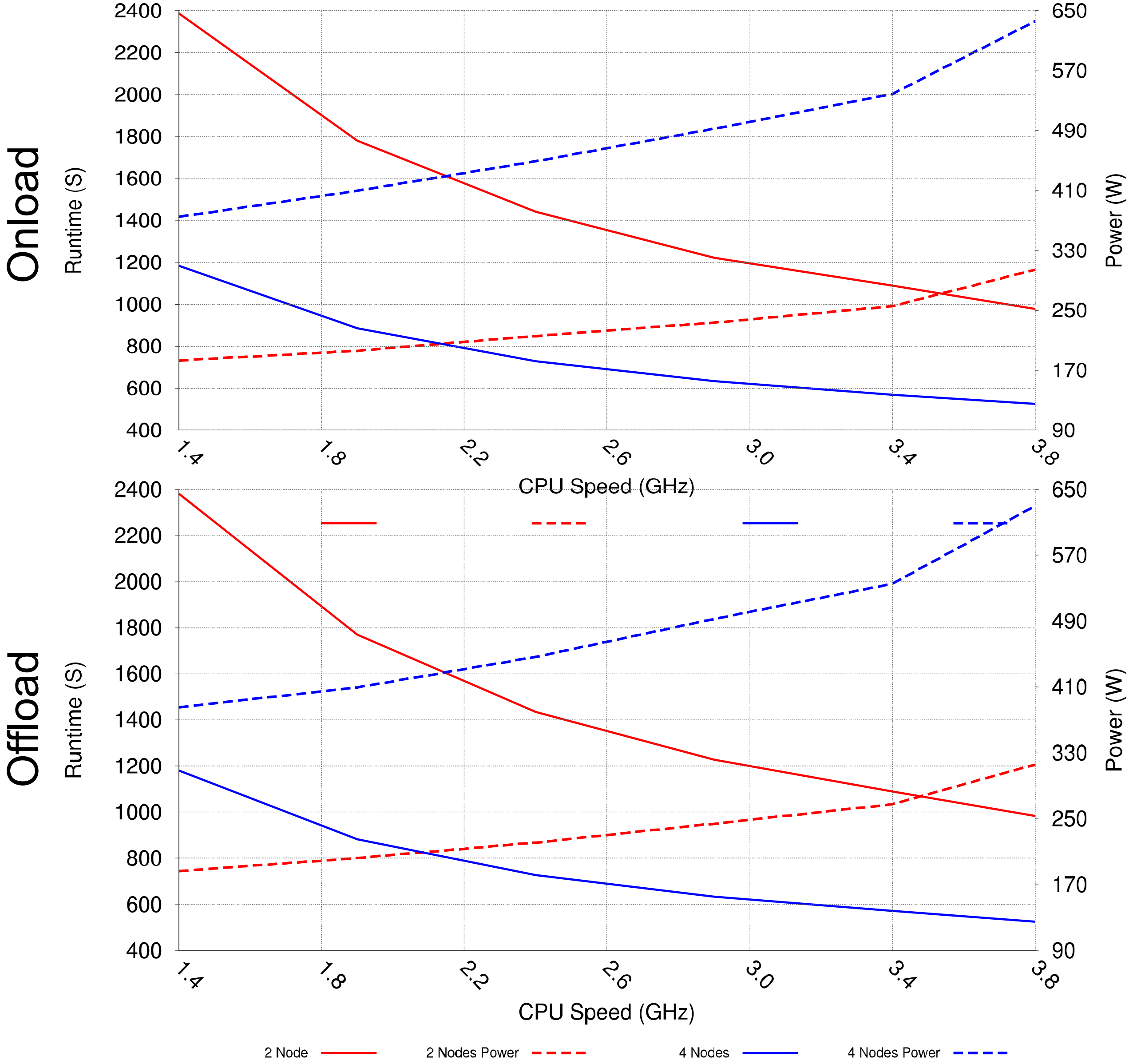
MILC Runtime With Power



MILC is an application that does a Lattice computation

The performance of MILC improves with offload, especially at lower frequencies and when more nodes are used

LULESH Runtime With Power



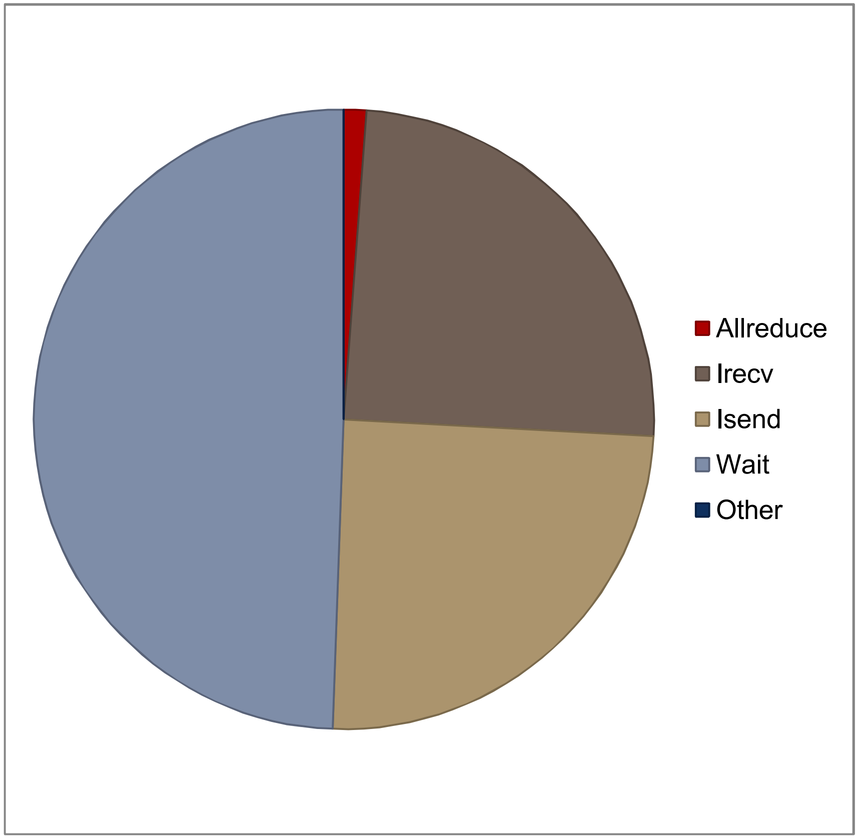
LULESH - hydrodynamics proxy app

Almost no impact, why?

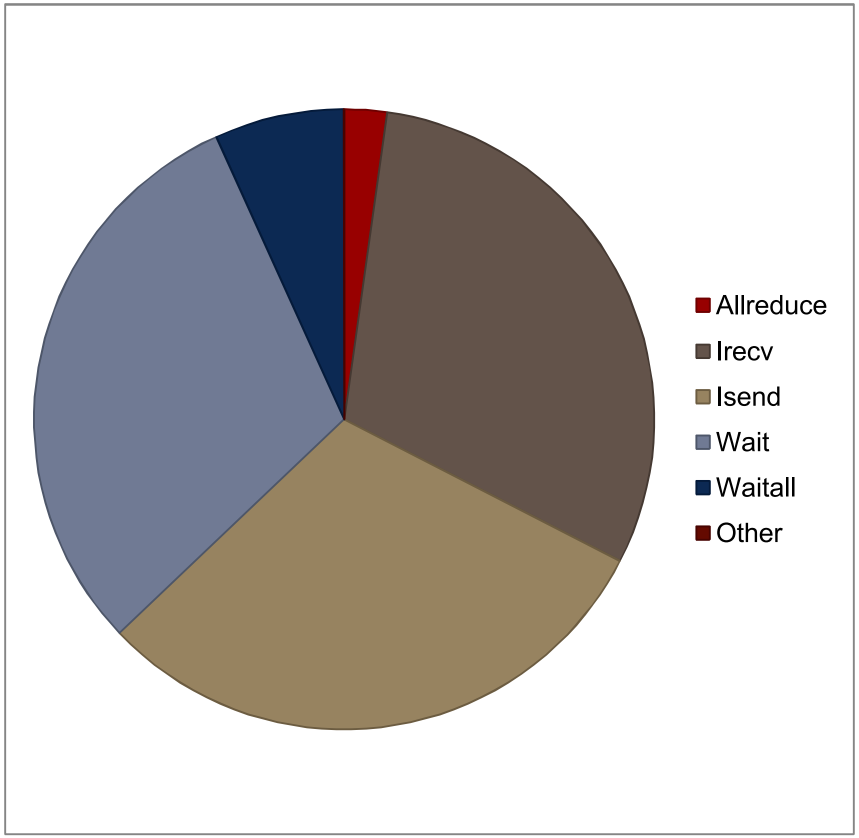
Distribution of MPI Calls

MILC Makes 17x more MPI calls per second than LULESH

MILC



LULESH



Summary

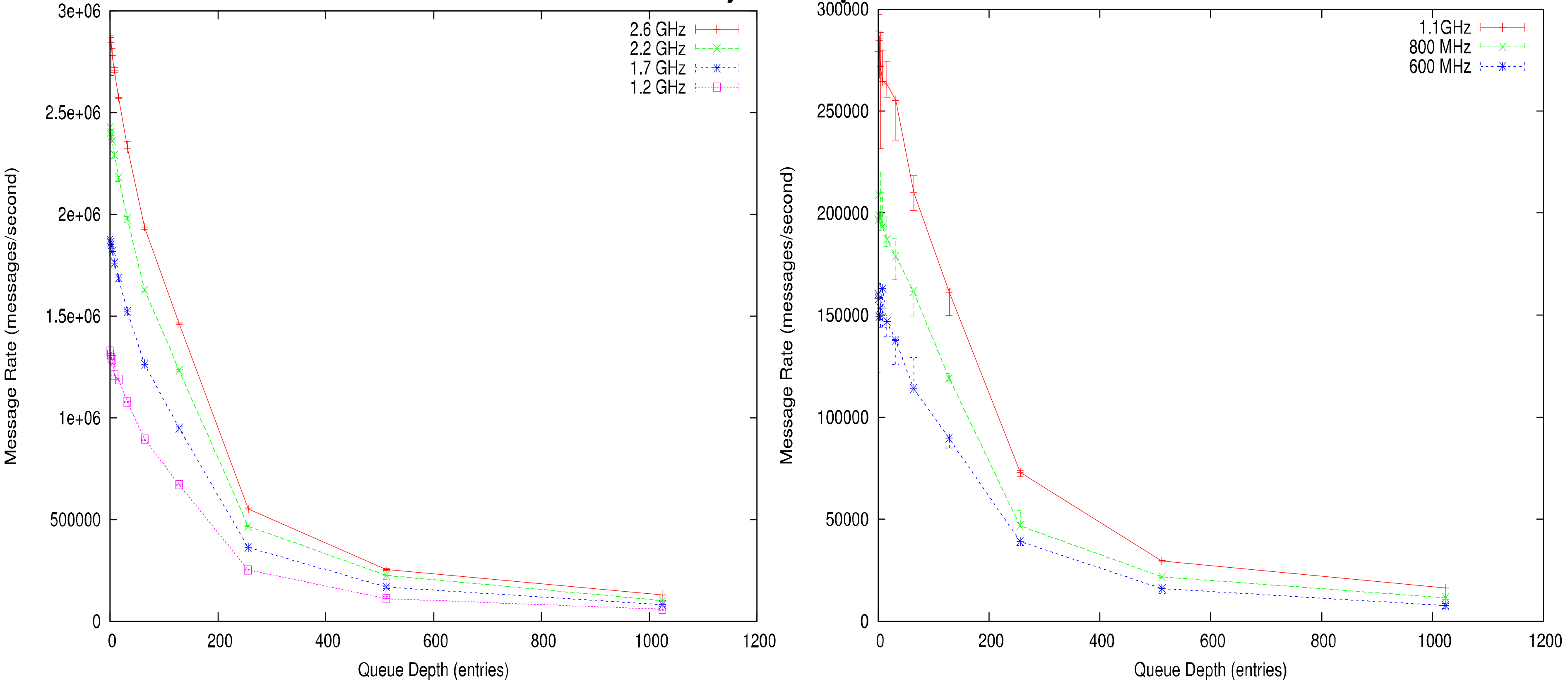
- Offload has advantages with slower cores
  - Bandwidth and latency advantages are clear

- Application impact is dependent on overall MPI usage
  - MILC sees 7-10% performance impact from onloaded scaling
  - LULESH not significantly impacted

- Expect that these results would be worse on many-core in absolute terms

Expect Worse Results

- Even with hardware support, and similar frequency many-cores are still dominated by deep out-of-order CPUs



(a) Sandy Bridge

(b) Xeon Phi

B. W. Barrett, R. Brightwell, R. E. Grant, S. D. Hammond, and K. S. Hemmert. An evaluation of MPI message rate on hybrid-core processors. *International Journal of High Performance Computing Applications*, 28(4):415–424, 2014.

Modeling Many-Core Matching

- Understanding how many cores to allocate to MPI matching for many-core architectures is critical
  - Heavy-weight vs. Light-weight cores have vastly different message rate matching capabilities
  - Modeling multiple cores used for MPI networking processing is novel

- Informing future design with respect to core allocation tradeoffs
  - Compute cores vs. networking cores

- Patrick G. Bridges, Matthew G.F. Dosanjh, Ryan E. Grant, Anthony Skjellum, Shane Farmer, Ron Brightwell, "Preparing for Exascale: modeling MPI for many-core systems using fine-grain queues", in Proceedings of the 3rd Workshop on Exascale MPI held in conjunction with the International Conference for High Performance Computing, Networking, Storage, and Analysis (Supercomputing 2015), Austin, TX, Nov. 15-20, 2015.

Modeling Many-Core Matching

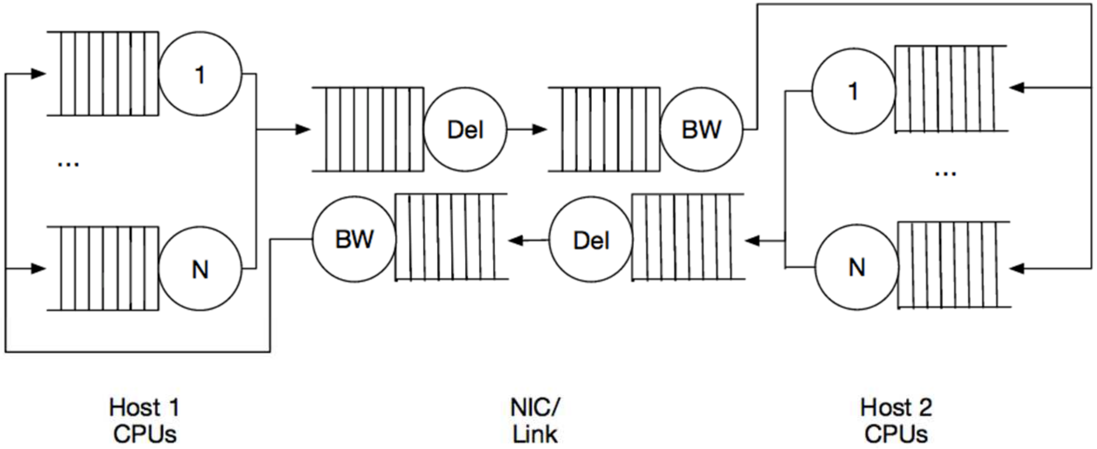
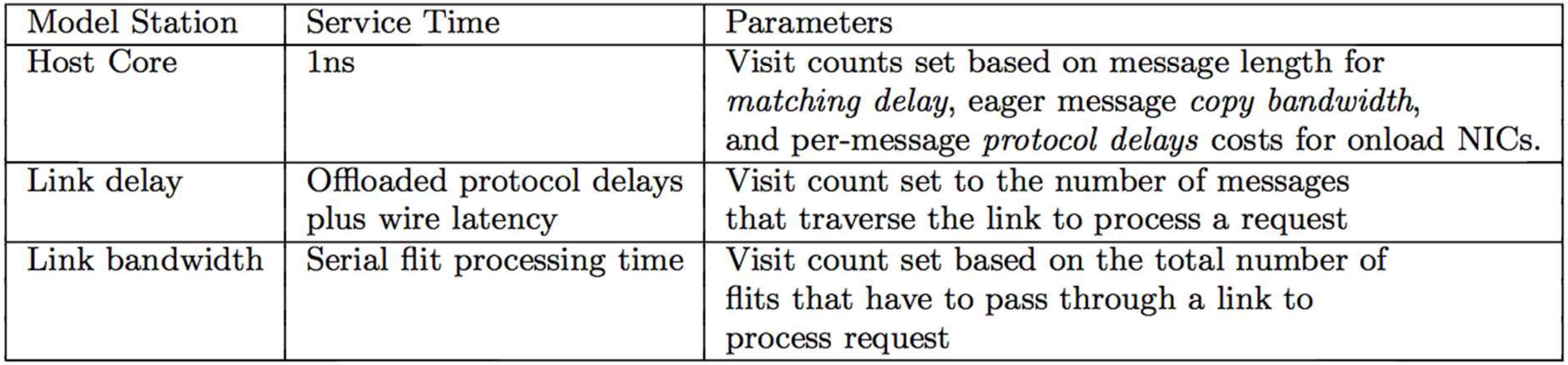


Figure 1: General Queueing Architecture of Model

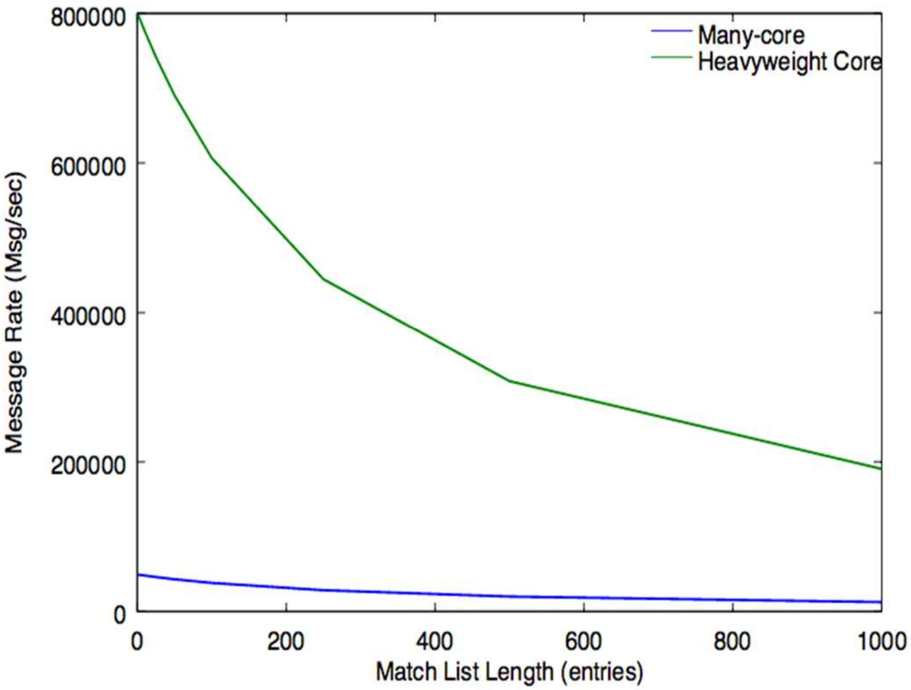


Model Station	Service Time	Parameters
Host Core	1ns	Visit counts set based on message length for *matching delay*, eager message *copy bandwidth*, and per-message *protocol delays* costs for onload NICs.
Link delay	Offloaded protocol delays plus wire latency	Visit count set to the number of messages that traverse the link to process a request
Link bandwidth	Serial flit processing time	Visit count set based on the total number of flits that have to pass through a link to process request

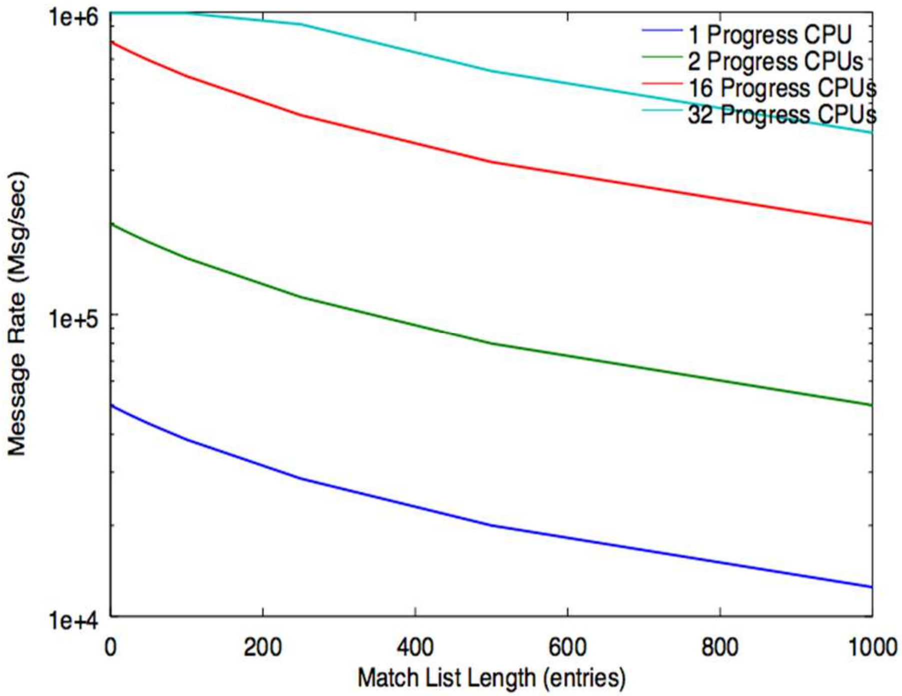
Modeling Many-Core Matching

- Model allows for many-core vs. heavy core comparisons
- Validated vs. ARM Cortex A9

Message Rate with Varying Match List Length on different core types



Aggregate Message Rate vs. Match List Length



Modeling Many-Core Matching

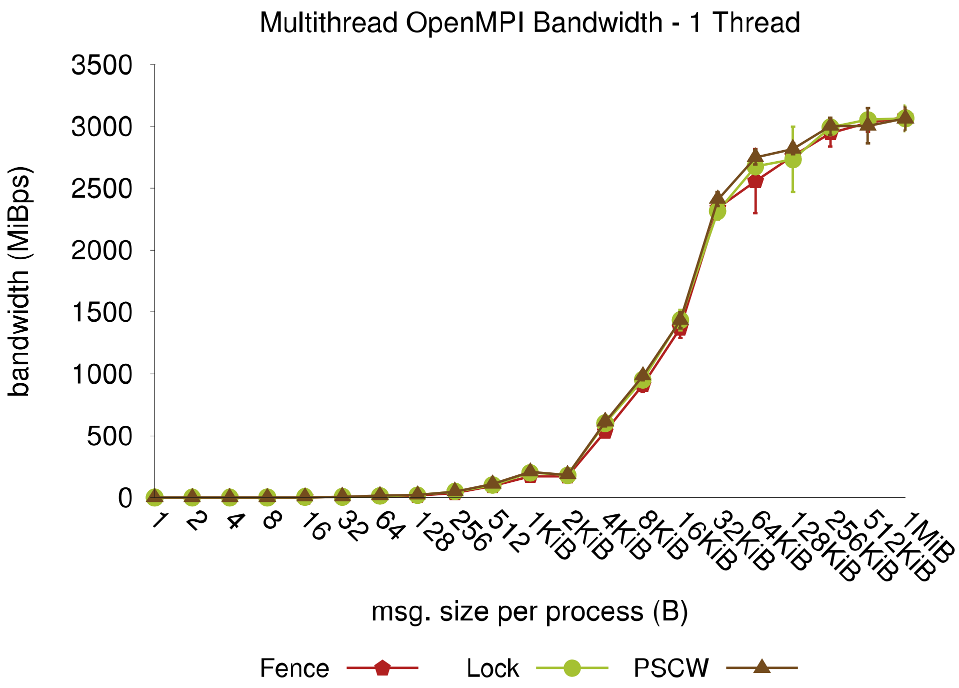
- Future Work
  - Investigate models for the Phi in more detail
  - Integrated NIC models
  - Large scale models with many nodes

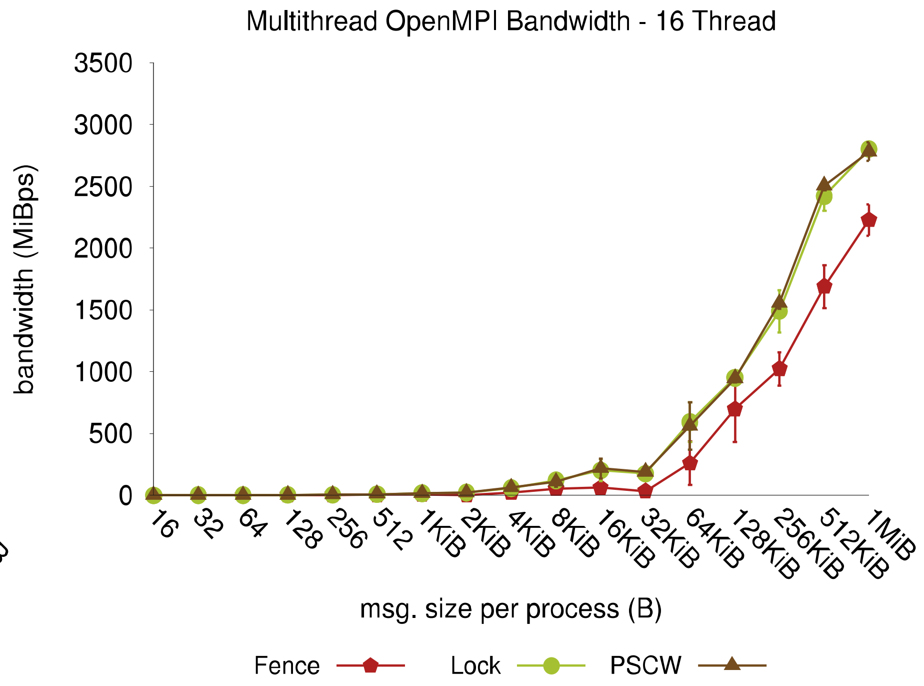
How well do alternative data movement techniques work?

- MPI One-sided communication (RMA)
  - Promising communication model for many-thread/task approaches
- Problems this work addresses:
  - Apps want to explore this communication models BUT….
    - No functionality tests exist with multiple concurrent threads (MT)
    - MPI code is not well tested
    - Performance is not well understood
    - Many options for memory window synchronization
  - What works and what doesn't?
  - What's the best performance option?
- Other Problem – How do we design algorithms to take full advantage of this communication mechanism?
- Matthew Dosanjh, Taylor Groves, Ryan E. Grant, Patrick G. Bridges, Ron Brightwell, "RMA-MT: A Benchmark Suite for Assessing MPI Multi-threaded RMA Performance", in Proceedings of the 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2016), Cartagena, Columbia, May 16-19, 2016. (Acceptance rate: 20%).

RMA-MT

- Need simple micro benchmarks for testing performance and functionality
- Need application proxies to to understand impacts on potential production codes


Multithread OpenMPI Bandwidth - 1 Thread


Multithread OpenMPI Bandwidth - 16 Thread

RMA-MT

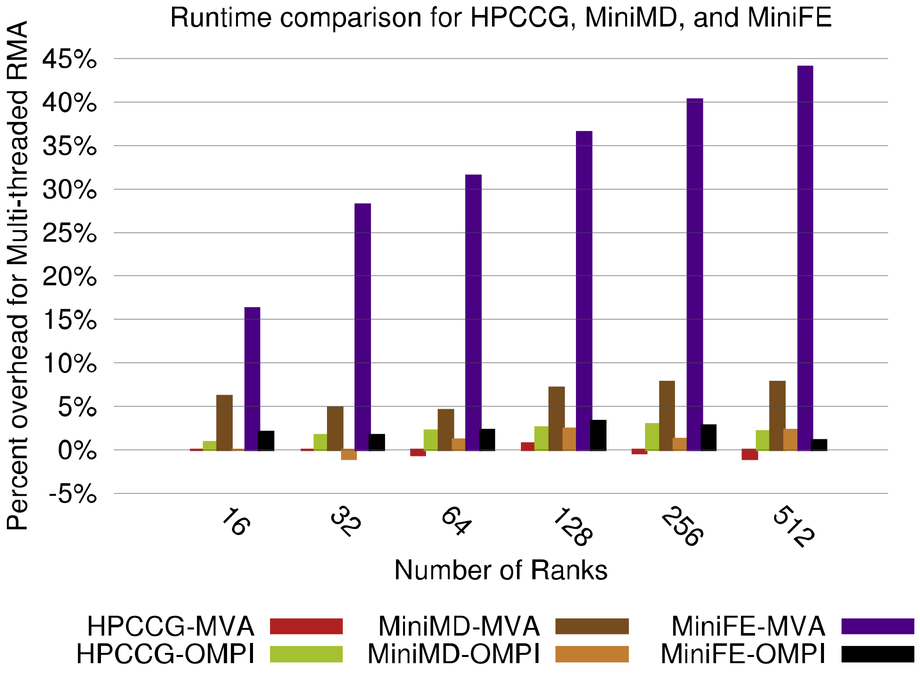
- Benchmarks were instructive
  - Found numerous bugs in MPI implementations
    - Segmentation Faults
    - Incorrect data transfers
    - Performance issues
    - Caught errors that could not be handled
  - Performance was instructive
    - Certain synchronization mechanisms that should be good choices were not
      - Optimizations are fixing these issues
  - Resulted in numerous pull requests on the OMPI source code to fix issues/performance

RMA-MT

- Microbenchmarks were re-written in a naïve manner to demonstrate performance of existing algorithms
  - The algorithms themselves should really be re-designed
    - Different area of research (that we're working on at SNL)
- Neglecting MiniFE-MVA

Performance is not that bad

Could be a lot better.



RMA-MT

- Future Work
  - Study performance on larger systems
    - 512 processes is not enough for full production understanding
  - Use a variety of system architectures
    - Need to understand performance/functionality for more MPI implementation code paths
  - Demonstrate scaling to large scale (100,000s of processes)
  - Currently work on a IEEE cluster paper with large systems
    - Trinity, Cielo, TACC Stampede and production clusters at SNL

What are possible ramifications to alternative data movement strategies?

- Remote Direct Memory Placement (RDMA) – underlying mechanism to MPI RMA

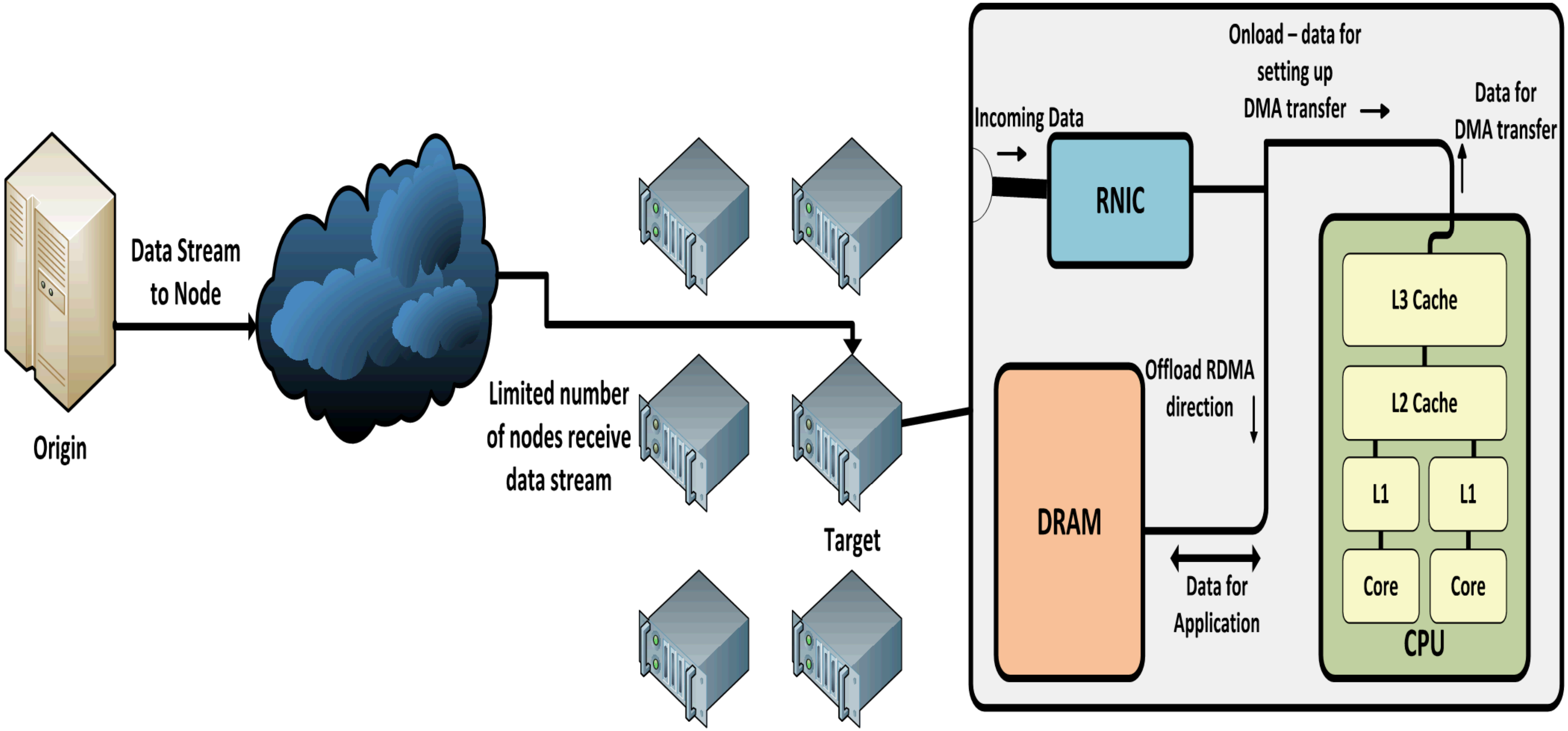
- Fundamentally RDMA is another actor on memory, requiring resources and creating contention

- Question: What impact does overlapped data movement have when the NIC is moving data during computation
  - Data may not be related to current computation (e.g. visualization or analytic data for post compute phase evaluation)
  - Cache injection strategies may be important here

Taylor Groves, Ryan E. Grant, Dorian Arnold, “NiMC: Characterizing and Eliminating Network-Induced Memory Contention”, in Proceedings of the 30th IEEE International Parallel & Distributed Processing Symposium (IPDPS), May 23-27, Chicago, IL. (Acceptance Rate: 23%).

Network Induced Memory Contention

- Data movement path



NiMC

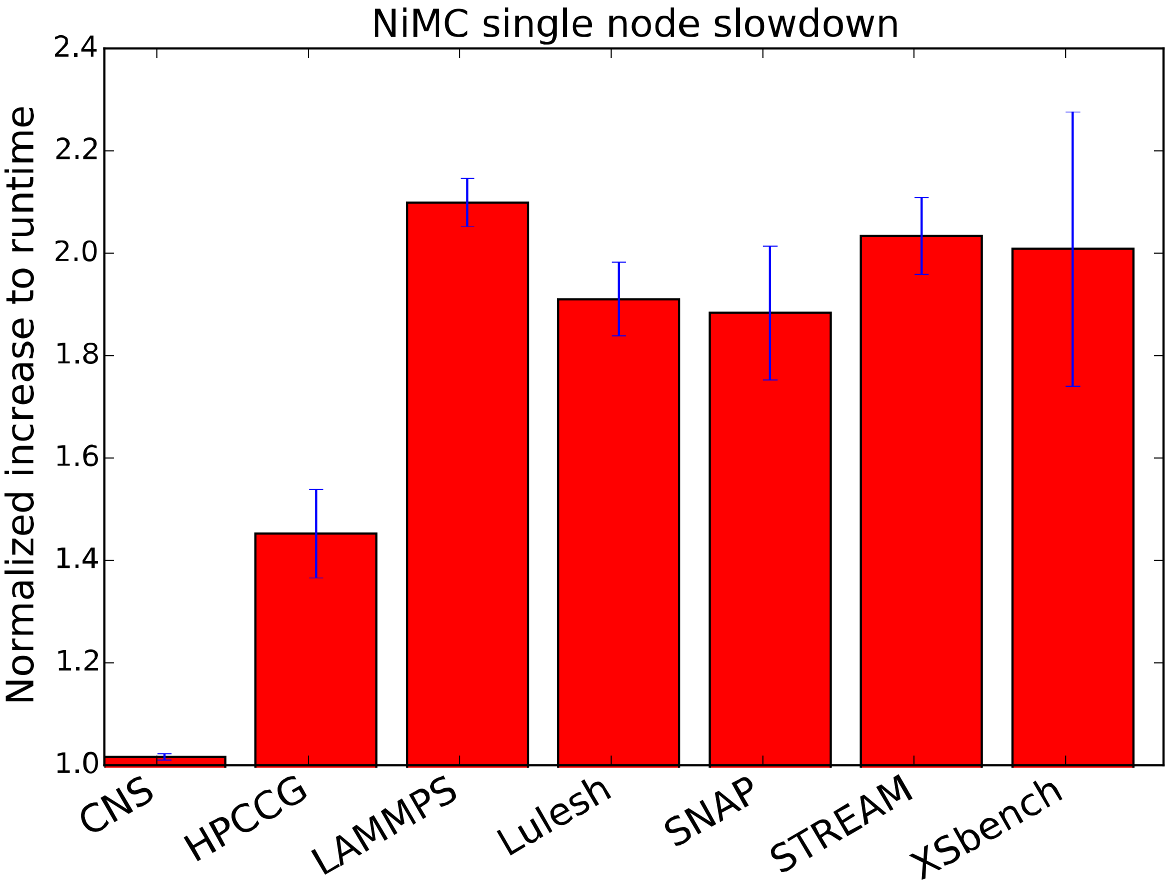
- Workloads:
  - Microbenchmark: STREAM (modified for first touch)
  - Proxy/mini apps: CNS, HPCCG, SNAP, LULESH, XSBench
  - Application: LAAMPS
- Platforms
  - Intel and AMD architectures going back 4 generations (Intel)
  - Variety of onload/offload InfiniBand NICs
  - Xeon Phi (KNC)
  - Large variation of memory bandwidths: 12.8-240 GB/s
  - Observed STREAM triad bandwidth with 51%-0% performance degradation

NiMC

- Impact on single node benchmarks (secondary node only sends data, not involved in communication)

Sandy bridge

--Onload FDR

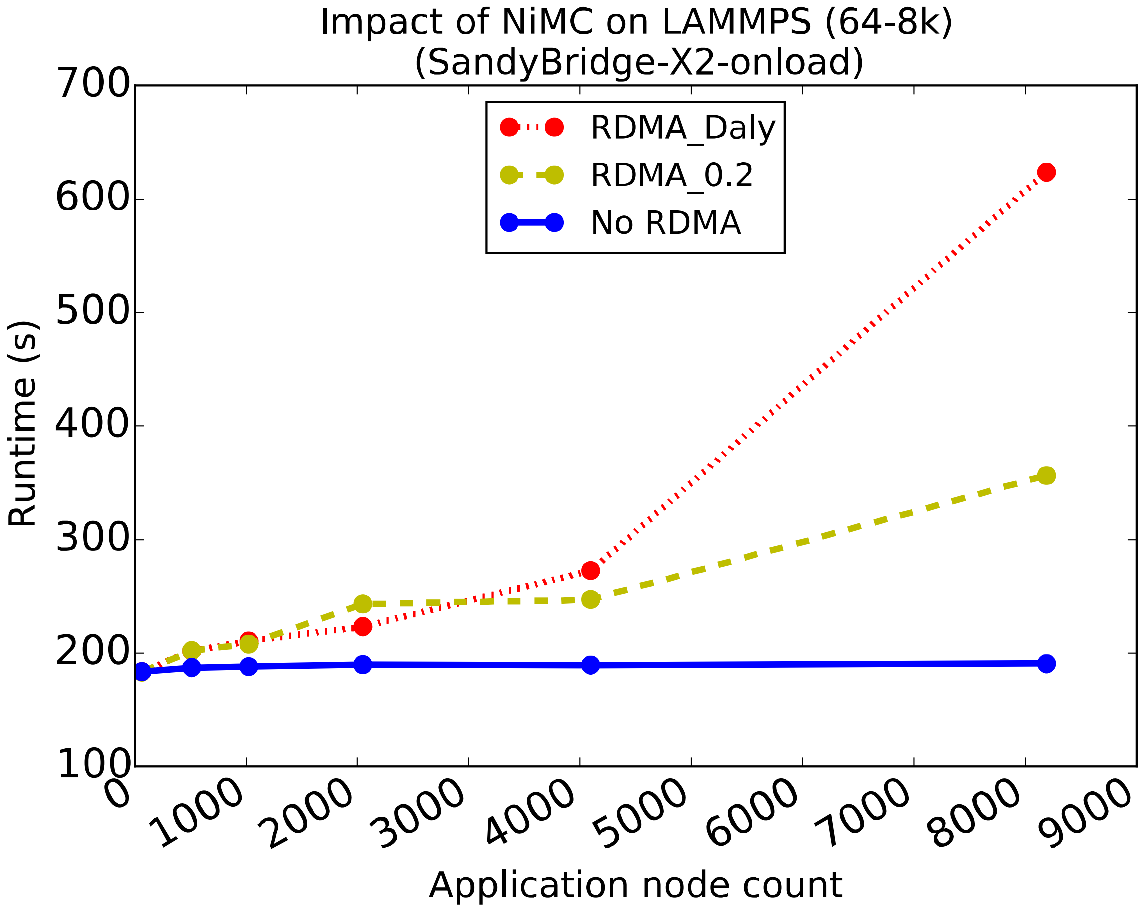


NiMC

- Application tests show large slowdown

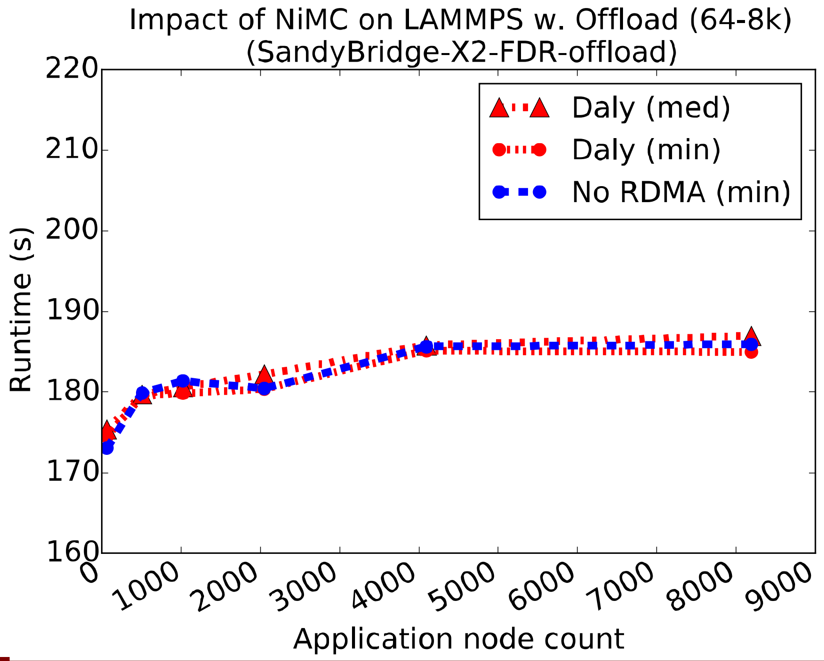
Assumes app never stops to use data
--Not something apps today do, but burst buffers/NVRAM storage may be used in this manner in the future

Could also represent sending data that is not needed in close time period to its reception

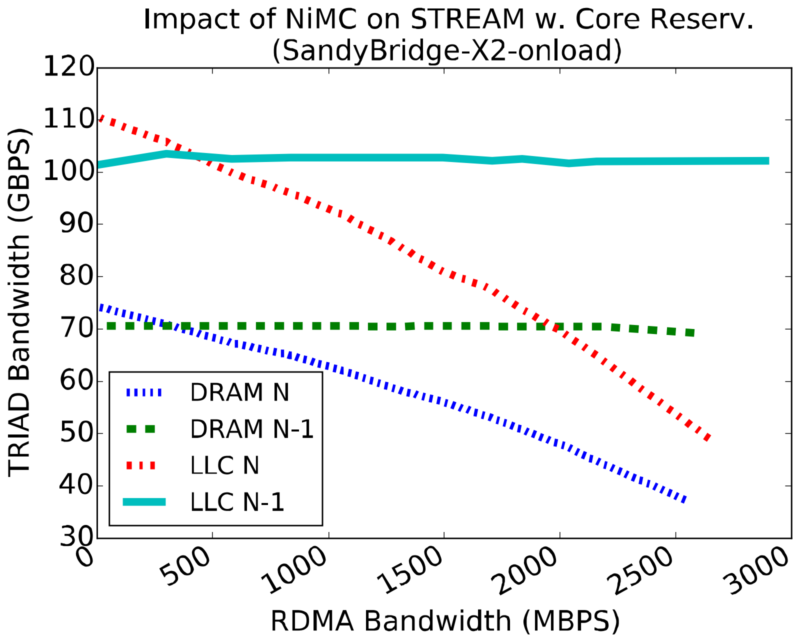


NiMC

- Multiple solutions to avoid this memory contention
- Can treat NiMC much like other minor disturbances in execution like OS noise
- Easiest solution, use offloaded networking cards (current generation only)



Impact of NiMC on LAMMPS w. Offload (64-8k)
(SandyBridge-X2-FDR-offload)



Impact of NiMC on STREAM w. Core Reserv.
(SandyBridge-X2-onload)

NiMC

- Status/Future plans:

- Talked to Intel about onloaded Qlogic HCA performance
  - Solutions coming for the use case/method used here

- Motivates further investigating cache injection from network devices
  - Results show that cache interference is a big problem

- If we get more cache injection control in the future, how do we use that to best advantage?

Exascale Computing

Extreme Scale Power Research

Power Research at SNL

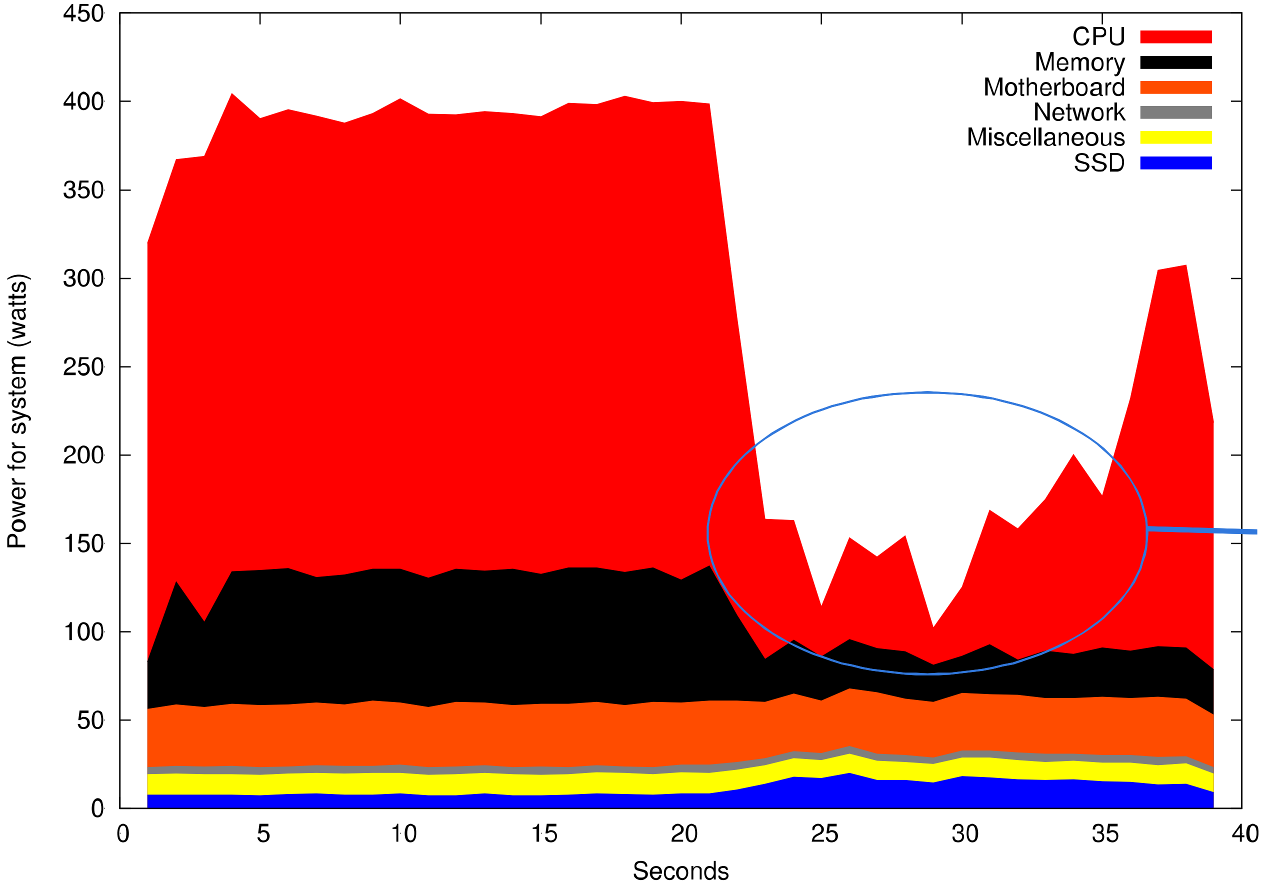
- Common Power control and monitoring interface
  - Power API – being used in ACES Trinity supercomputer
  - Developed at SNL with community input (including UNM)

- Power API has progressed alongside exploratory research

- Research answering questions like:
  - Are there opportunities during I/O to save power (like checkpoints)?
  - What ramifications are there to power/energy saving runtime approaches in the context of large systems (reliability)?
  - How do program optimizations impact power consumption on different architectures?

Saving Power During Checkpointing

- Application starts checkpointing and wants to manage power



Power saving opportunity

Source: Mills, Bryan; Grant, Ryan E; Ferreira, Kurt B; Riesen, Rolf, "Evaluating energy savings for checkpoint/restart", in Proceedings of the 1st International Workshop on Energy Efficient Supercomputing, 2013

Power for Exascale Computing

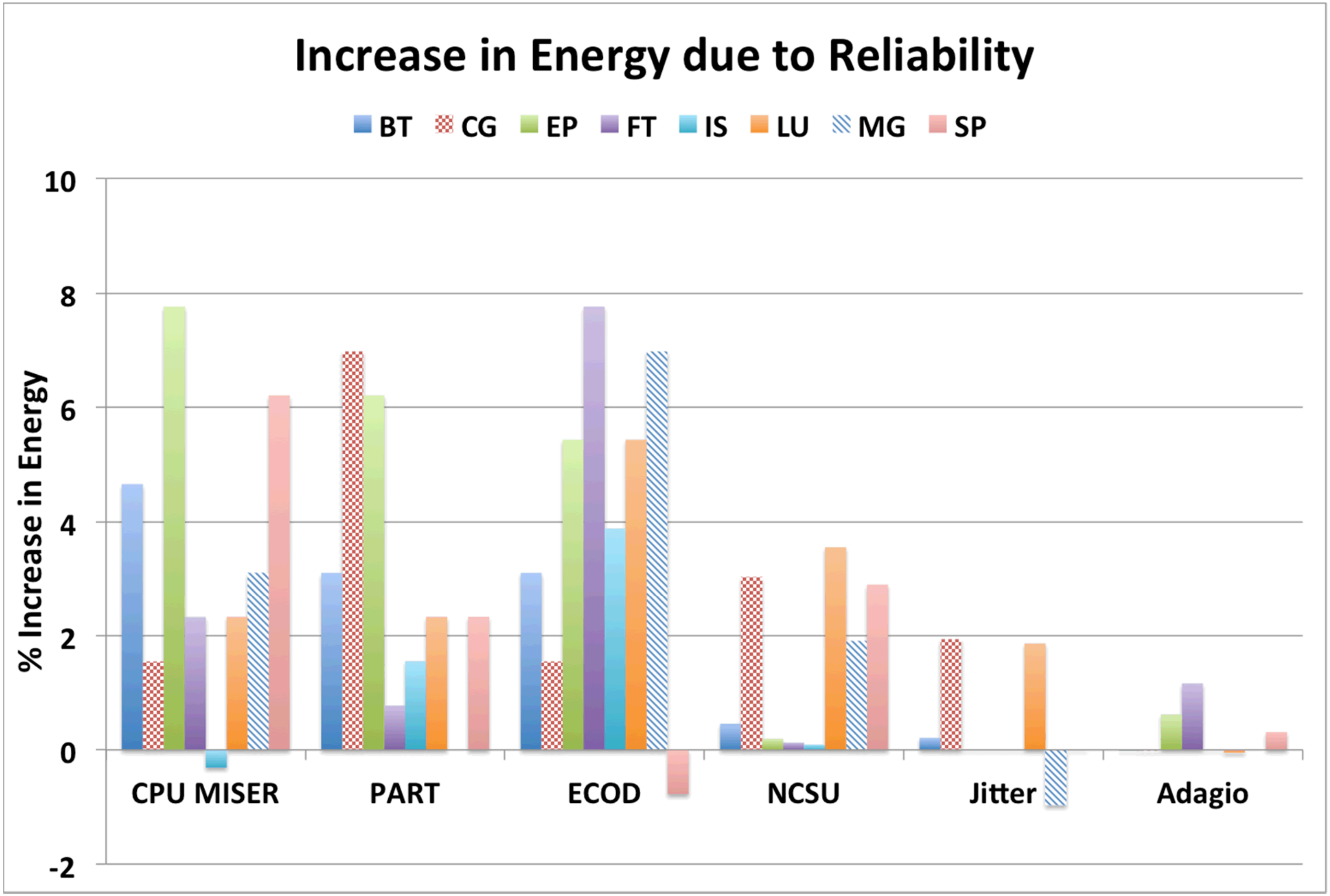
What ramifications are there to power/energy saving runtime approaches in the context of large systems (reliability)?

Retrospective on Power Saving Techniques

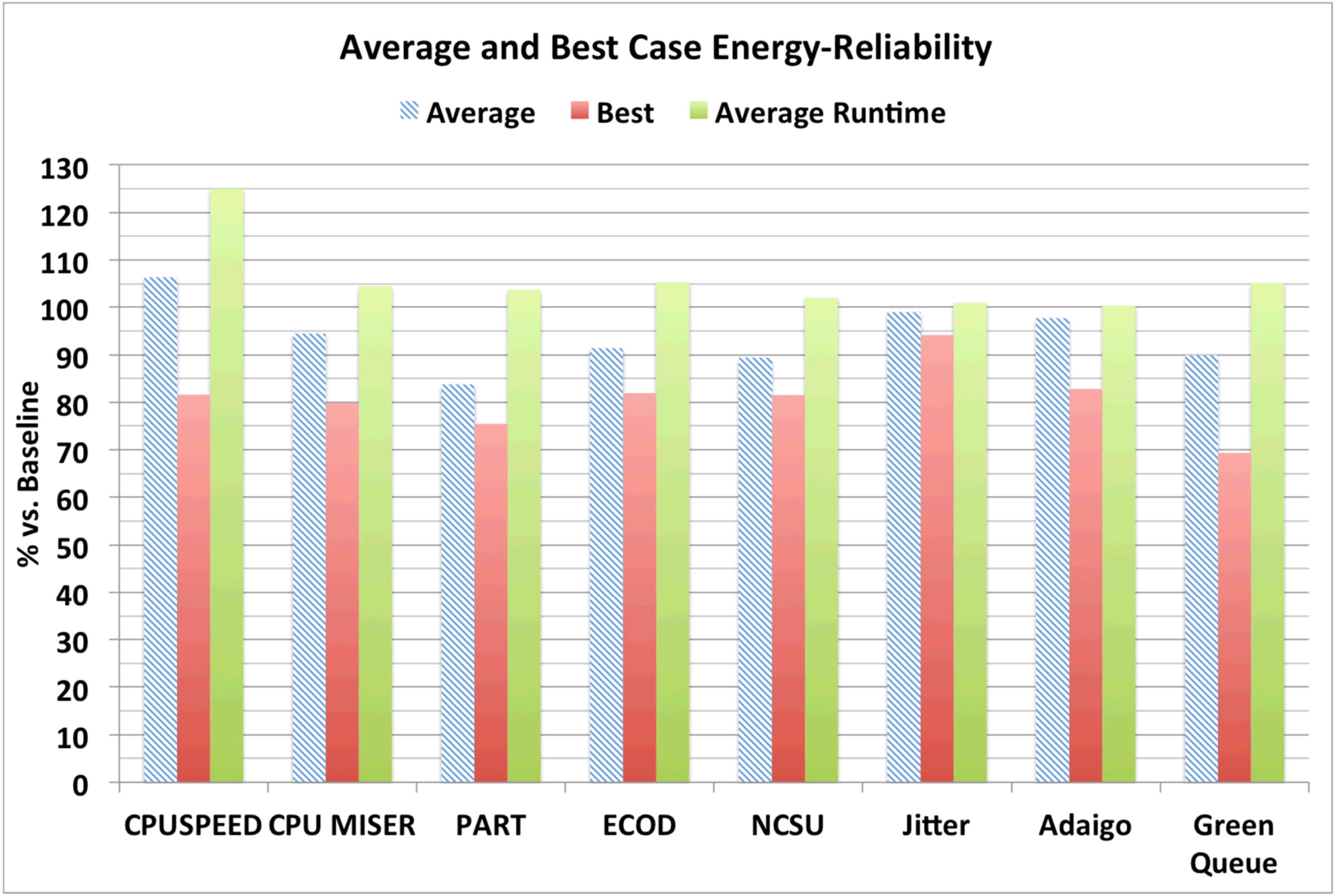
- Analyze history of energy saving methods
  - NAS benchmarks – common comparison point
- Study:
  - CPUSpeed (2005)
  - CPU Miser (2007)
  - PART (2005)
  - ECOD (2009)
  - NCSU method from 2006
  - Jitter (2005)
  - Adagio (2009)
  - Green Queue (2012)

Ryan E. Grant, Stephen Olivier, James Laros, Allan Porterfield, and Ron Brightwell, “Metrics for Evaluating Energy Saving Techniques for Resilient HPC Systems”, 10th Workshop on High-Performance, Power-Aware Computing (HP-PAC 2014), Held in conjunction with the International Parallel and Distributed Processing Symposium (IPDPS 2014), Phoenix, AZ, May 19-23, 2014.

Analysis of Techniques



Analysis of Techniques



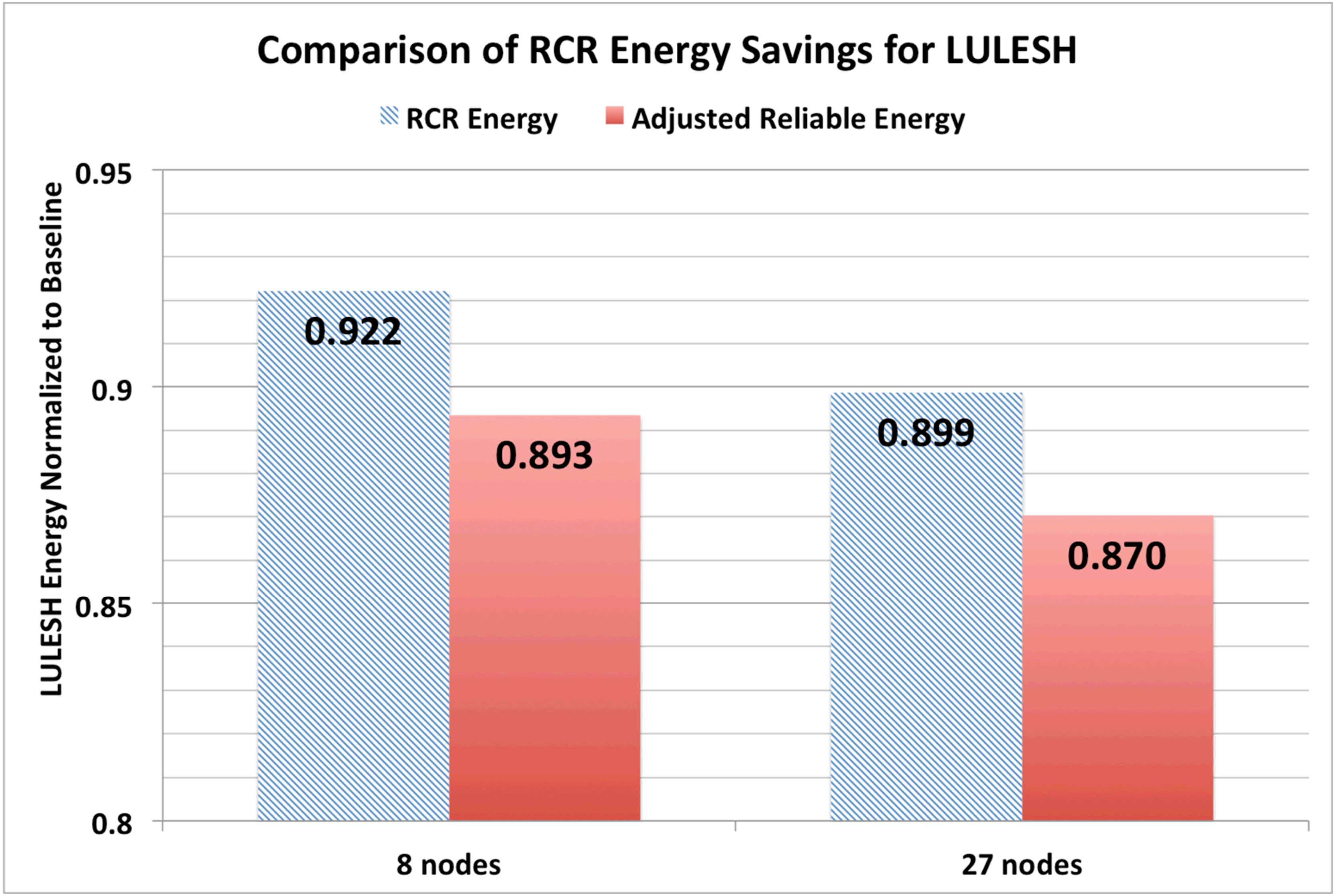
Analysis of Techniques

- Results show that techniques are still quite helpful
- The most aggressive energy saving techniques suffer the most from the reliability adjustment
  - Have longer runtimes in exchange for greater energy savings
  - Increases probability of failures happening during run
- Some techniques have applications that benefit from the addition of reliability concerns
- Benefits of running at lower temperatures not explored

Case Study

- Studied MAESTRO/RCR energy-efficiency techniques
  - Detects memory bandwidth saturation and reduces thread concurrency
  - Scales back processor frequency on some threads to reduce memory pressure
- Results improve by considering reliability as runtimes are improved
- Overall, a 2.9% average improvement in energy savings numbers due to reliability

Exascale Computing



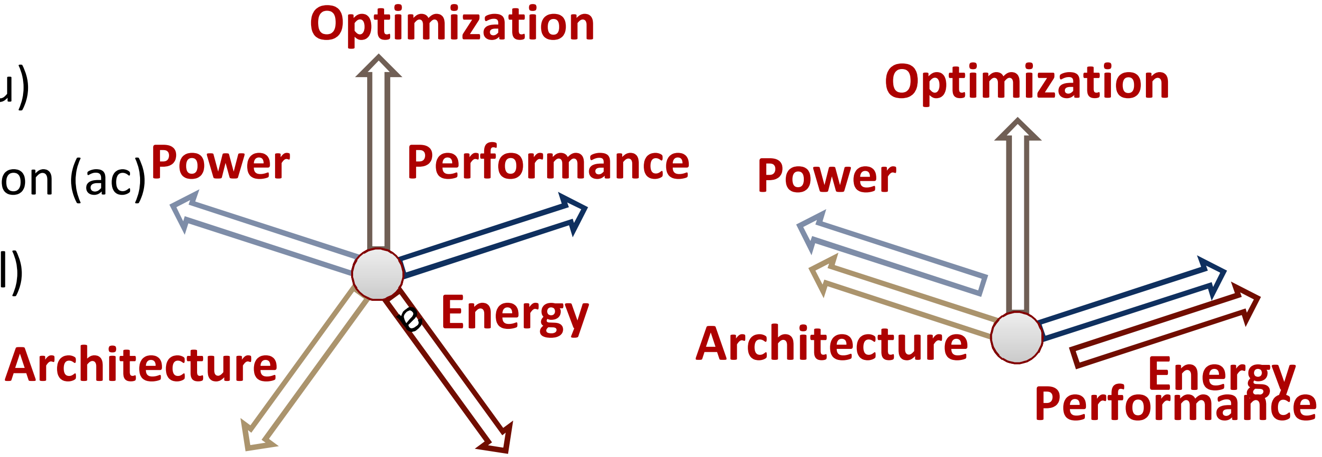
Power for Exascale Computing

How do program optimizations impact power consumption on different architectures?

Investigating architecture, power, and performance relations

- Explicit shock hydrodynamics
  - LULESH
- *Optimizations*
  - Loop fusion (fu)
  - Global allocation (ac)
  - Data layout (dl)
- Metrics
  - Time to solution
  - Power
  - Energy

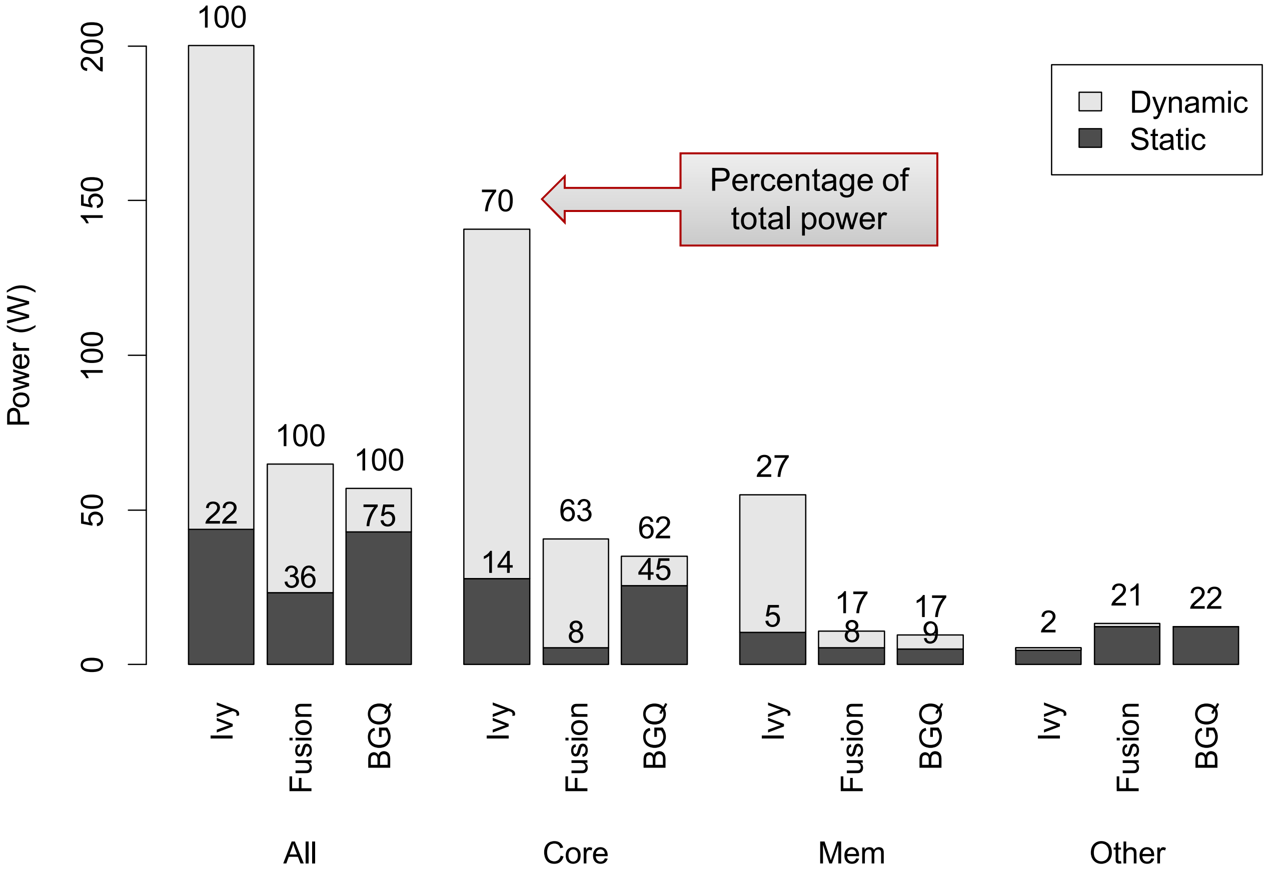
- Architecture class
  - Enterprise server - Intel Xeon
  - Low cost consumer - AMD Fusion
  - Low power HPC - IBM BG/Q



Edgar A. Leon, Ian Karlin, **Ryan E. Grant**, “Optimizing Explicit Hydrodynamics for Power, Energy, and Performance”, IEEE International Conference on Cluster Computing (Cluster 2015), Chicago, IL, USA, September 8-11, 2015. (Acceptance Rate: 24%). **Best Paper Award Winner**
Original slides courtesy Edgar Leon, LLNL

Power per node at a glance: LULESH NoOpt

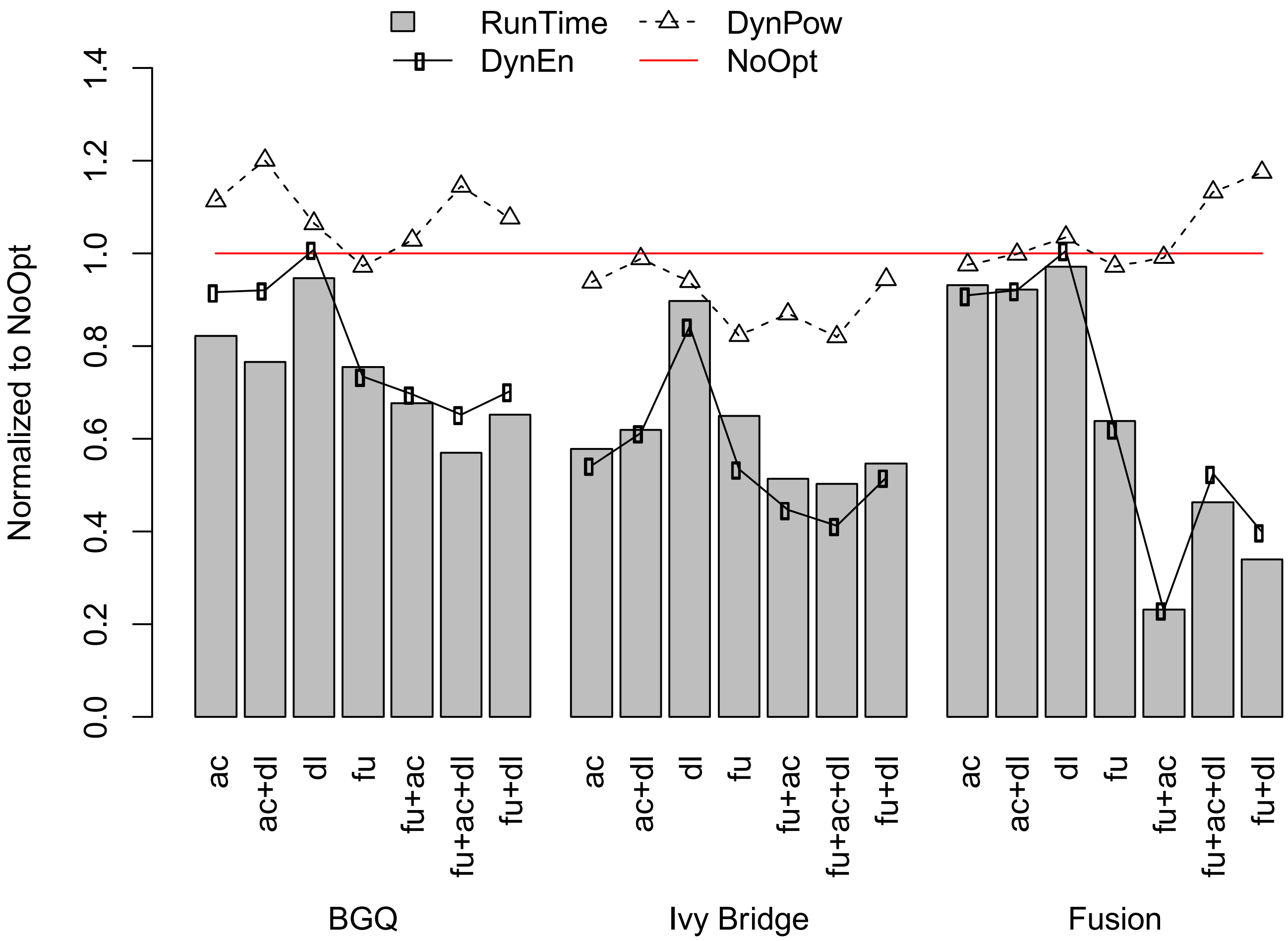
- Dynamic range
  - Higher on x86
  - Component specific
- Ivy: Significantly higher draw
- BG/Q: High static power
- BG/Q, Fusion: Similar memory ratio
- Dynamic power focus
  - Optimizations
  - Future systems



Percentage of total power

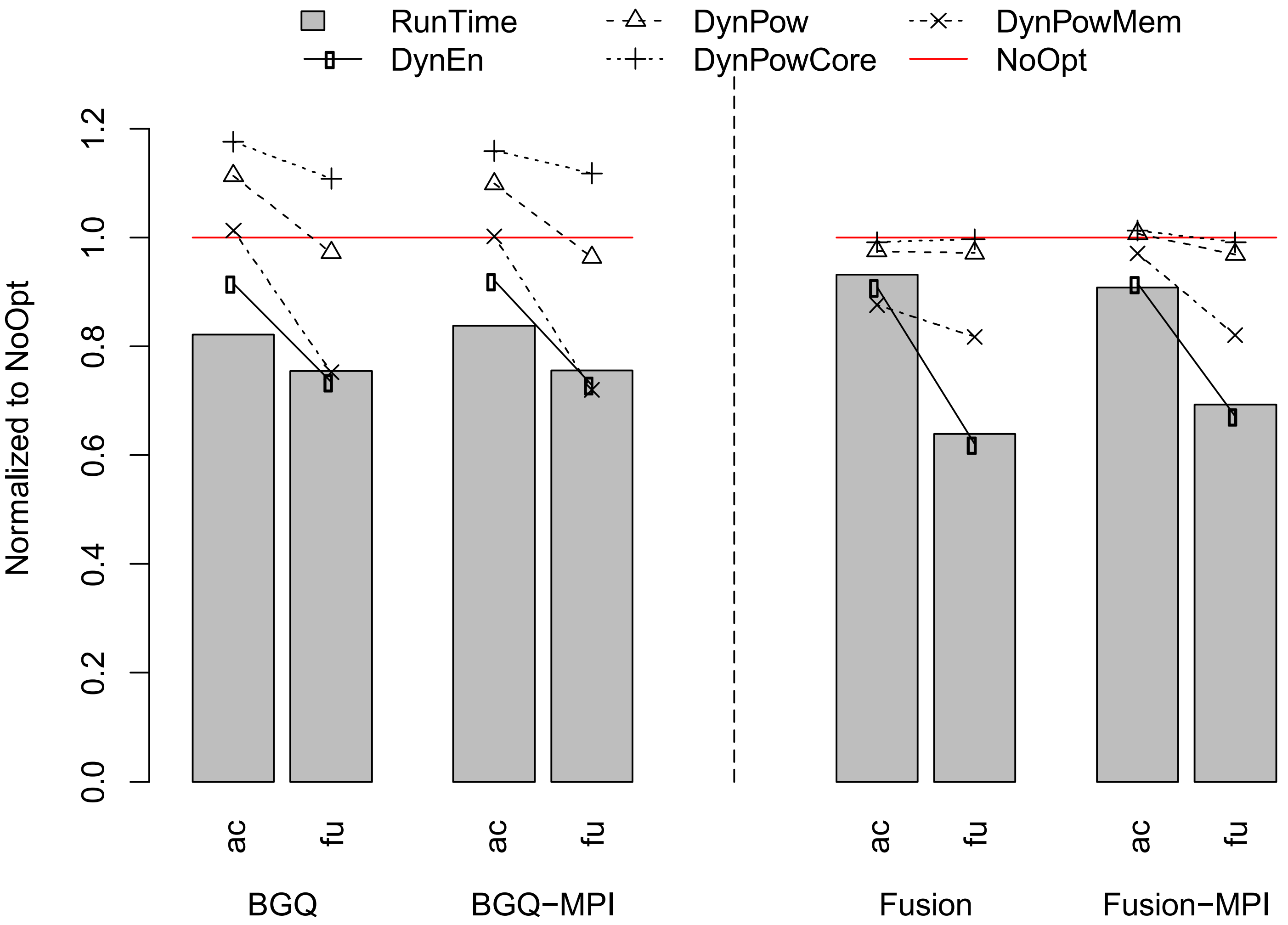
Optimizations: Improved runtime and energy, power within 20%

- Data normalized to NoOpt
- Energy follows runtime
- BG/Q and Ivy share best runtime
  - fu+ac+dl
- BG/Q and Fusion share best power
  - fu
- No single best for all

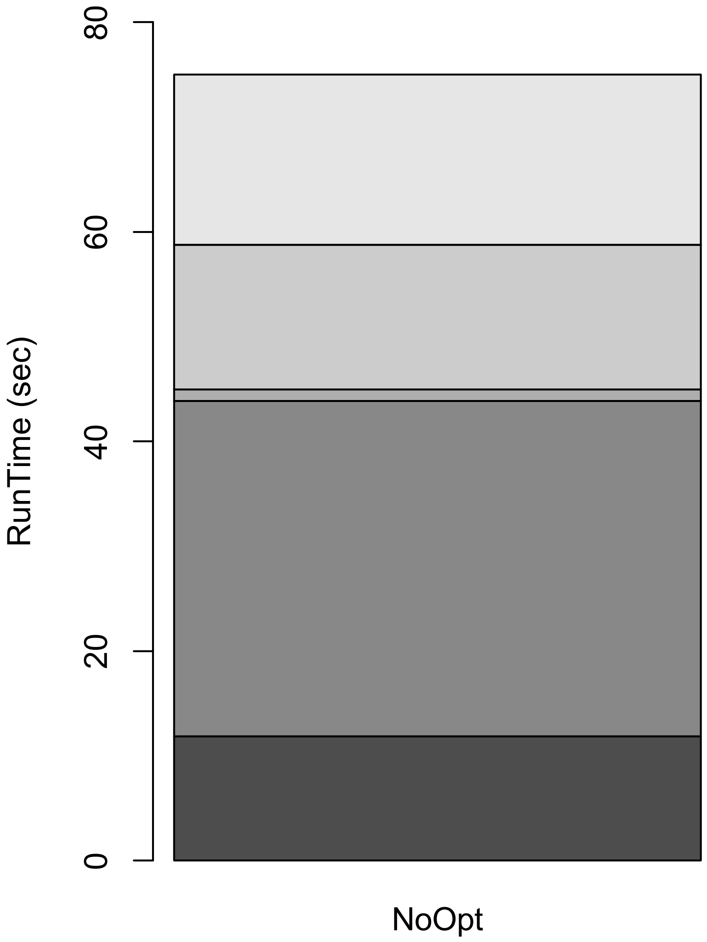


MPI analysis and verification

Results from optimizations on single nodes carried over to node counts up to 64



Zooming in:
5 code regions from application developers view

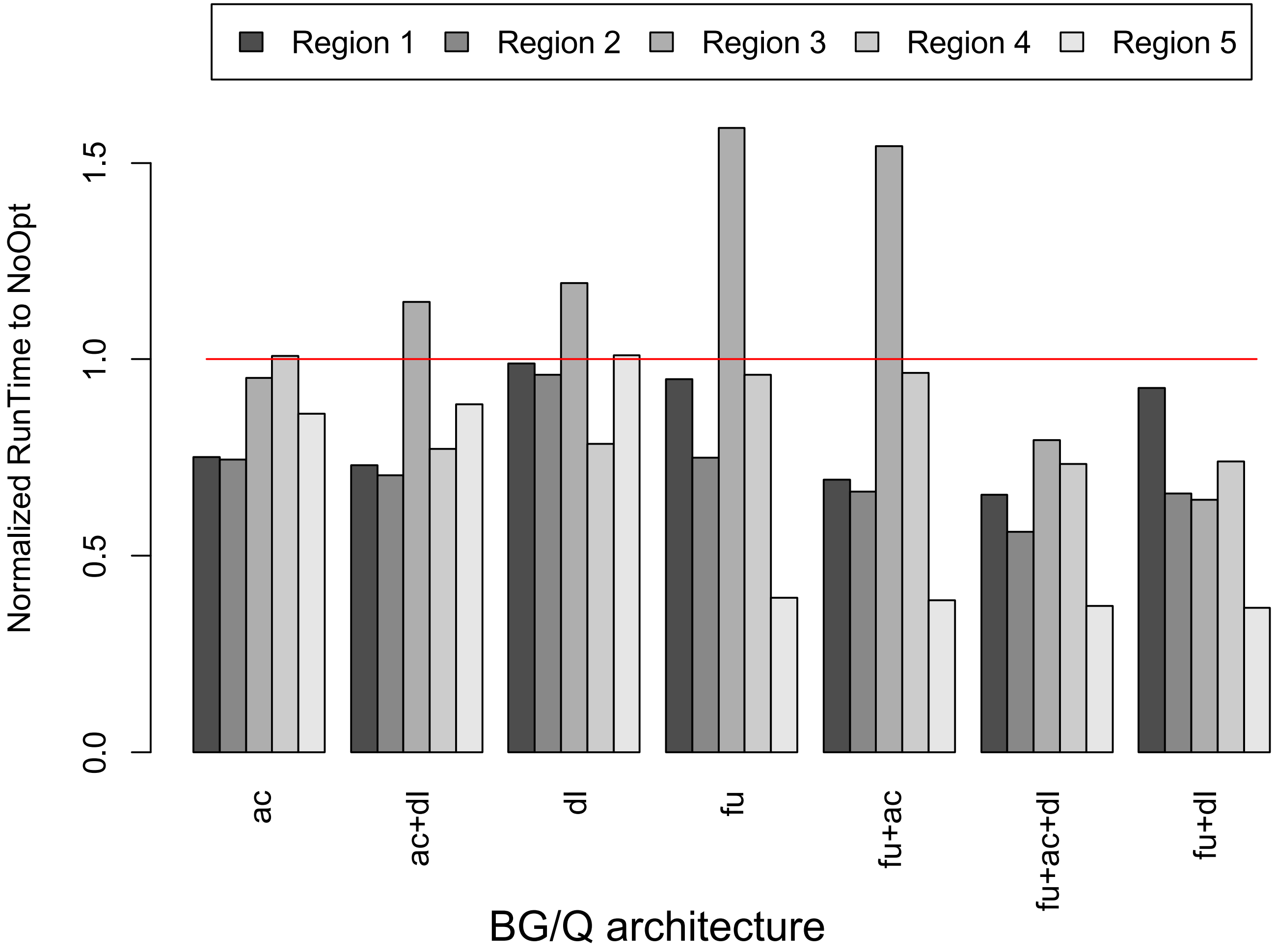


Significant differences between regions

BG/Q	INT %	FPU %	IPC	L1H %	BW GB/s	Time %
R5	62.4	37.6	0.321	77.2	13.72	21.62
R4	45.5	54.5	0.654	88.4	9.12	18.41
R3	74.1	25.9	0.216	75.7	20.88	1.46
R2	62.5	37.5	0.554	93.6	15.56	42.71
R1	47.4	52.6	0.541	90.4	18.38	15.80

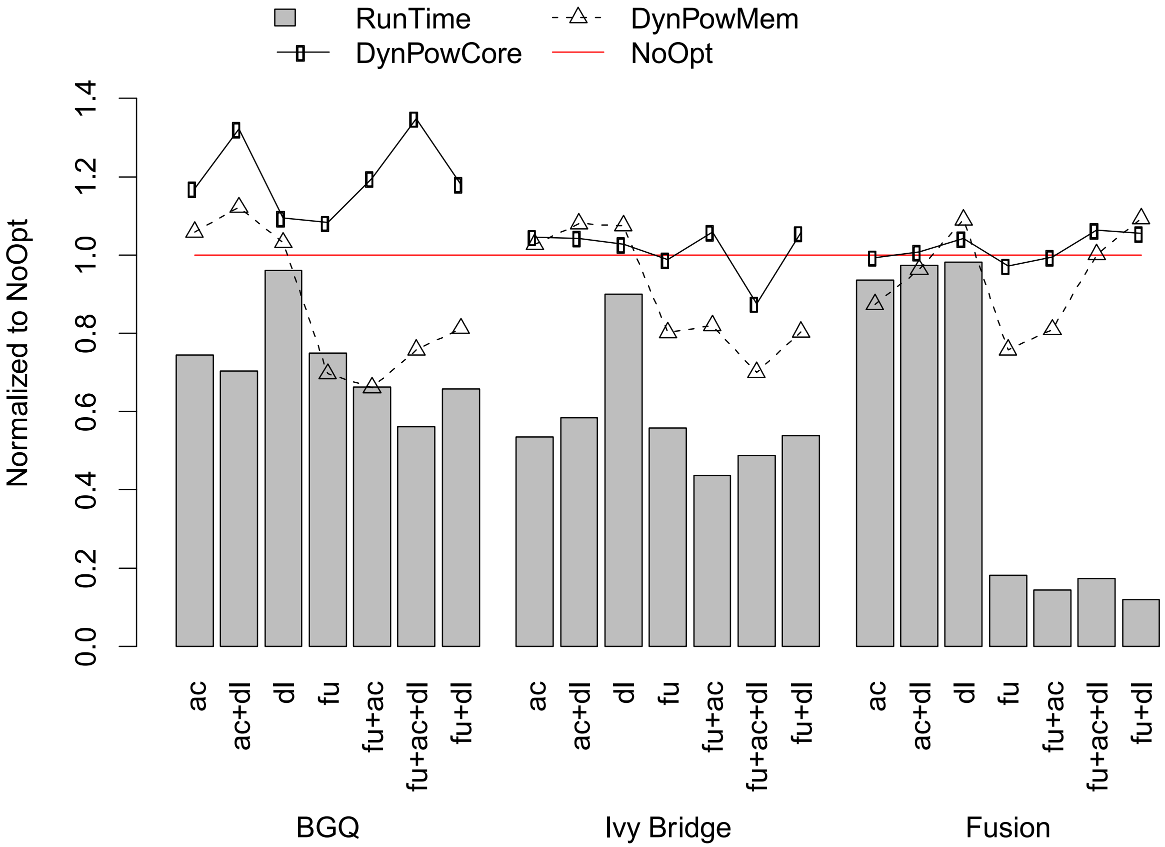
Effect of optimizations is region dependent

- Optimizations benefit most regions
- Region 3 not happy
- Region 5 favors fu
- Optimal varies by region
- Optimization combinations often increase performance



Region 2: loop fusion effective across architectures

- Improved runtime
- Less data motion–memory power
- More compute intensity–core power
- BG/Q: memory to compute bound



	BG/Q	Ivy Bridge	Fusion
Counter	*Inst. Exec.*	*L2 Misses*	*L1 Misses*
Pearson	0.974	0.969	0.752

BG/Q	INT%	FPU%	*IPC*	L1H%	*BW*
NoOpt	62.5	37.5	*0.554*	93.6	*15.56*
Best	60.1	39.9	*0.924*	96.8	*10.33*

Best optimizations for power, energy, and performance

- Overall “best” applies per-region optimizations
- Largest region dominates only on BG/Q
- Energy follows runtime
- Core and memory power optimizations often differ
- Memory power best with fu

Best optimizations

		R1	R2	R3	R4	R5	Prog
BG/Q	PCore	NoOpt	NoOpt	fu+ac	dl	NoOpt	NoOpt
	PMem	fu+dl	fu+ac	dl	fu	fu+ac	fu+ac
	Power	fu+dl	fu	fu	NoOpt	fu	fu
	Energy	fu+ac+dl	fu+ac+dl	fu+dl	dl	fu	fu+ac+dl
	Time	fu+ac+dl	fu+ac+dl	fu+dl	fu+ac+dl	fu+dl	fu+ac+dl
Ivy Bridge	PCore	dl	fu+ac+dl	NoOpt	dl	fu	NoOpt
	PMem	fu	fu+ac+dl	fu+dl	fu+ac+dl	fu+ac	fu+dl
	Power	fu	fu+ac+dl	fu	fu+ac+dl	fu	fu+ac+dl
	Energy	ac	fu+ac+dl	fu	fu+ac+dl	fu+ac	fu+ac+dl
	Time	ac	fu+ac	fu+ac+dl	ac+dl	fu+dl	fu+ac+dl
Fusion	PCore	fu+ac	fu	ac+dl	fu+ac	fu+ac	fu+ac
	PMem	fu+ac	fu	fu+dl	fu	fu	fu
	Power	fu+ac	fu	ac+dl	ac	ac	fu
	Energy	fu+ac	fu+dl	ac+dl	fu+ac	fu+ac	fu+ac
	Time	fu+ac	fu+dl	dl	fu+ac	fu+ac	fu+ac

Optimizations such as fu help all objectives across architectures

SNL/UNM Collaborations

Accepted/Published Papers

- Taylor Groves, Ryan E. Grant, Dorian Arnold, “NiMC: Characterizing and Eliminating Network-Induced Memory Contention”, in Proceedings of the 30th IEEE International Parallel & Distributed Processing Symposium (IPDPS), May 23-27, Chicago, IL. (Acceptance Rate: 23%).
- Matthew Dosanjh, Taylor Groves, Ryan E. Grant, Patrick G. Bridges, Ron Brightwell, “RMA-MT: A Benchmark Suite for Assessing MPI Multi-threaded RMA Performance”, in Proceedings of the 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2016), Cartagena, Columbia, May 16-19, 2016. (Acceptance rate: 20%).
- Patrick G. Bridges, Matthew G.F. Dosanjh, Ryan E. Grant, Anthony Skjellum, Shane Farmer, Ron Brightwell, “Preparing for Exascale: modeling MPI for many-core systems using fine-grain queues”, in Proceedings of the 3rd Workshop on Exascale MPI held in conjunction with the International Conference for High Performance Computing, Networking, Storage, and Analysis (Supercomputing 2015), Austin, TX, Nov. 15-20, 2015.
- Matthew Dosanjh, Ryan E. Grant, Patrick Bridges, Ron Brightwell, “Re-evaluating Network Onload vs. Offload for the Many-Core Era”, IEEE International Conference on Cluster Computing (Cluster 2015), Chicago, IL, USA, September 8-11, 2015. (Acceptance Rate: 24%).

Work In Progress

- Edgar Leon, Ian Karlin, Ryan E. Grant, Matthew Dosanjh, “Program Optimizations: The Interplay Between Power, Performance, and Energy”, submitted to Journal of Parallel Computing.
- Matthew Dosanjh, Taylor Groves, Ryan E. Grant, Patrick G. Bridges, Ron Brightwell, “Assessing MPI RMA-MT performance on Large-Scale Systems”, submitting to Cluster 2016.
- Taylor Groves, Ryan E. Grant, Scott Hemmert, Simon Hammond, Michael Levenhagen, Dorian Arnold, “SAI: Stalled, Active and Idle, Characterizing Next-Gen Networks”, submitting to Cluster 2016
- Taylor Groves, Ryan E. Grant, Aaron Gonzales, Dorian Arnold, “NiMC: Characterizing and Eliminating Network-Induced Memory Contention”, to be submitted to IEEE Transactions on Parallel and Distributed Systems

Teaching Outreach

- **Ryan E. Grant**, Stephen Olivier, “Networks and MPI for Cluster Computing”, National Science Foundation/IEEE Technical Council on Parallel Processing CDER Book Project on Parallel and Distributed Computing Topics for Undergraduate Core Courses. 1st Edition, Morgan Kaufmann, ISBN: 9780128038994, Pages: 360. Acceptance rate: 23.3%.

Future Outlook

- Will we have Exascale machines by 2020? mid 2020s?

- Failure of a single element could doom all of the others
  - Underperformance could be compensated for by overachievement elsewhere

- Easy scaling is no longer reality

- Success means opening up completely new areas for computational science

Final Thoughts

- Quantum computing could help solve some specific problems much faster than we can today

- New architectures could alter the current rate of improvement of memory

- Accelerator/Many-core architectures could be better leveraged for a larger variety of applications

- Do we need a technology transition akin to the one from vacuum tubes to transistors/integrated circuits?

Thank you

Questions?

Why Threading?

- MPI use cases continue to evolve
  - MPI+X implies the use of threads, e.g. OpenMP
- Potentially thousands of MPI processes on a single node (without threading)
  - Can complicate network resource management
- Sources of concurrency
  - Many core architectures
    - KNL
  - Increasing core counts
    - Traditional CPU design continues to add cores in new generations
  - OpenMP
    - Can we use MPI inside of OpenMP parallel regions?
  - New runtimes
    - Tasking runtimes potentially introduce many more concurrent uses of the MPI library

Living in a World with Threads

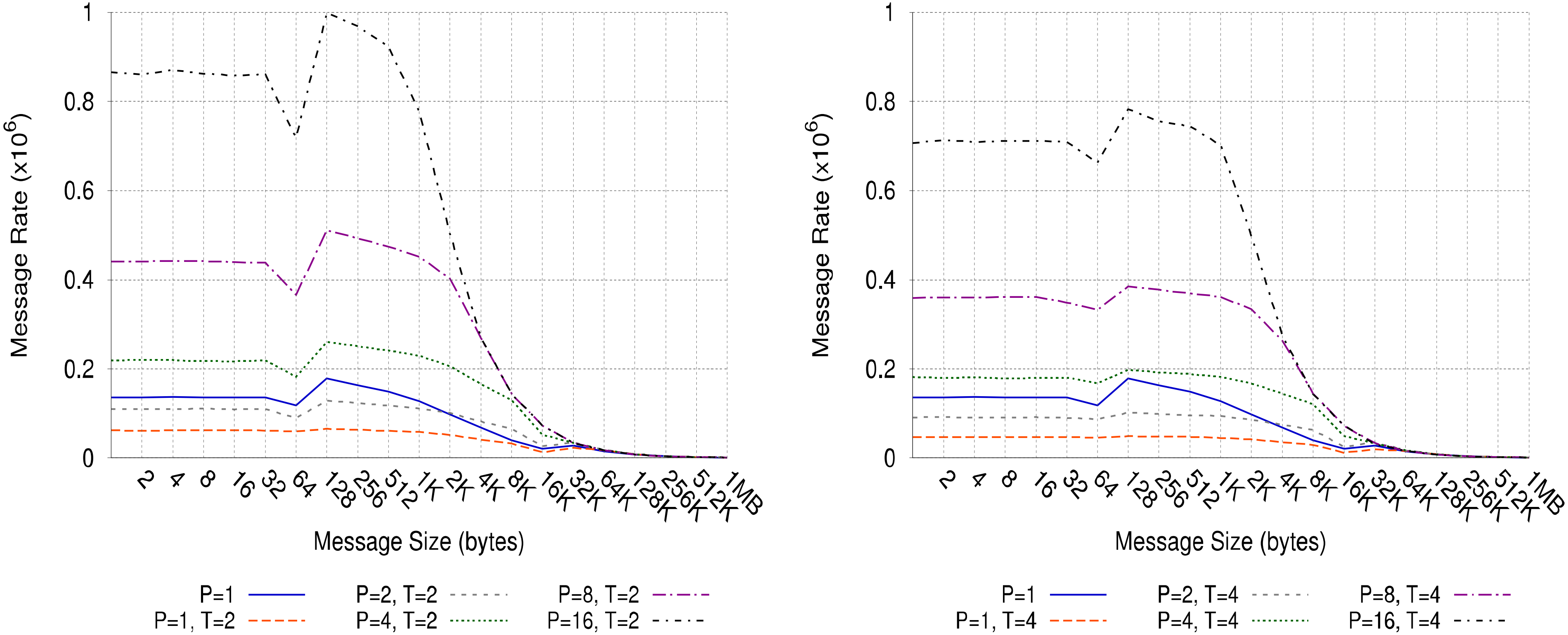
Desirable/required features:

- Low overhead
- Similar semantics to existing threading (minimal changes)
  - Ease of programmability
- Each thread has access to the communication library (no funneling)
  - Also ease programmability, e.g. use MPI calls in an OpenMP region
- Communication endpoint granularity matched to the work
  - Not too fine, not too coarse, just right…
  - Fine granularity at endpoints requires networking resources
    - Keeping track of many ranks, caching state related to these ranks, etc.

The problem with threads

- Threads introduce significant issues with concurrency in existing MPI implementations
  - MPI_THREAD_MULTIPLE is hard and implementations don’t do it well
  - Difficult to support concurrency without encountering conflicts
- Fixing MPI concurrency issues is hard, so multiple approaches proposed:
  - 1) Expose the parallelism (Endpoints)
    - Allow individual threads to have ranks in MPI (rank space explosion)
    - Exposes lots of complexity, thread addressability on the network
    - General solution that is applicable to many problems
  - 2) Allow for parallelism in a way that doesn’t require MPI to handle it
    - New concept: threads can contribute independently to a larger communication operation inside of MPI, doesn’t require the same synchronization methods

Exposing Parallelism in MPI



Impact of communication concurrency on message rate per second for P processes and T threads per process on an Intel Phi

Takeaway: Fewer threads = better performance, due to poor MPI multi-threading

James Dinan, Ryan E. Grant, Pavan Balaji, Dave Goodell, Douglas Miller, Marc Snir, Rajeev Thakur, "Enabling communication concurrency through flexible MPI Endpoints", International Journal of High Performance Computing Applications, Volume 28, Issue 4, pp. 390-405, November 2014. Impact factor: 1.63

Allow for Better Parallelism in MPI

- Concept of many actors (threads) contributing to a larger operation in MPI
- E.G. many threads work together to assemble a message
  - MPI only has to manage knowing when completion happens
  - These are actor/action counts, not thread level collectives, to better enable tasking models
- No heavy MPI thread concurrency handling required
  - Leave the placement/management of the data to the user
  - Knowledge required: number of workers, which is easily available
- Match well with Portals NIC capabilities
  - Use counters for sending/receiving
  - Utilize triggered operations to offload sends to the NIC

Persistent Partitioned Buffers

- Expose the “ownership” of a buffer as a shared to MPI
- Need to describe the operation to be performed before contributing buffer segments
- MPI implementation doesn’t have to care about sharing
  - Only needs to understand how many times it will be called before the operation can be completed
- Threads are required to manage their own buffer ownership such that the buffer is valid
  - The same as would be done today for code that has many threads working on a dataset (that’s not a reduction)
- Result: MPI is thread agnostic with a minimal synchronization overhead (atomic_inc)
  - Can alternatively use task model instead of threads

Example

- Like persistent communications, setup the operation

MPI_Partitioned_Send_Create(Comm, to_rank, to_tag, base_address, data_type, count, num_ contributors, &request)

- Start the request

MPI_ Start(request)

- Add items to the buffer

#omp parallel for …

MPI_Partitioned_ Add_to_buffer(MPI_Request *request, const void *in_data_loc, int count, MPI_Datatype type)

- Wait on completion

MPI_Wait(request)

- Optional: Use the same partitioned send over again

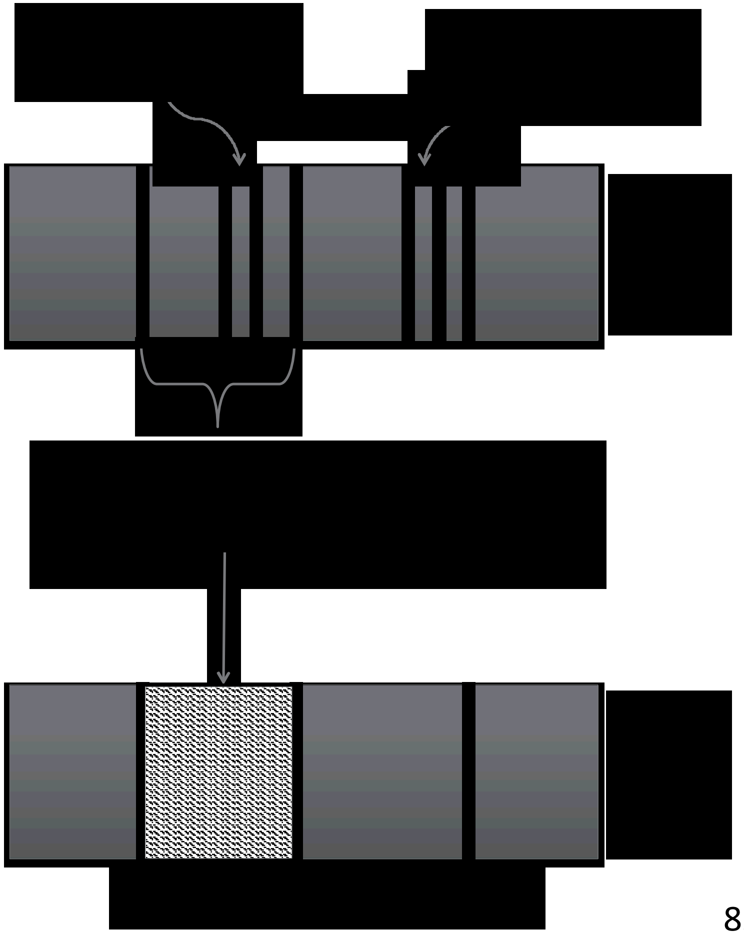
MPI_ Start(request)

Opportunities for Optimization

- MPI implementations can optimize data transfer under the covers
- Subdivide larger buffers and send data when ready
- Could be optimized to specific networks (MTU size)
- Number of messages will be:

  1 < #messages ≤ #threads/tasks

  For a partition with 1 part per thread
- Reduces the total number of messages sent, decreasing matching overheads



Different Approach to Threading

- Partitioned buffer operations can always be consider as multi-threaded
- Using partitioned sends doesn’t necessarily require locking in other parts of the MPI library
- Technically, using partitioned buffers would work with MPI_THREAD_SINGLE
  - Violates the definition of thread modes in the standard
  - If only using partitioned sends, no need for locking in the library
- Not part of this work but…
  - Could examine alternative threading modes for MPI, where user management of data is guaranteed and only inherently thread safe operations are called