Meeting the Communication Needs of Scalable Applications

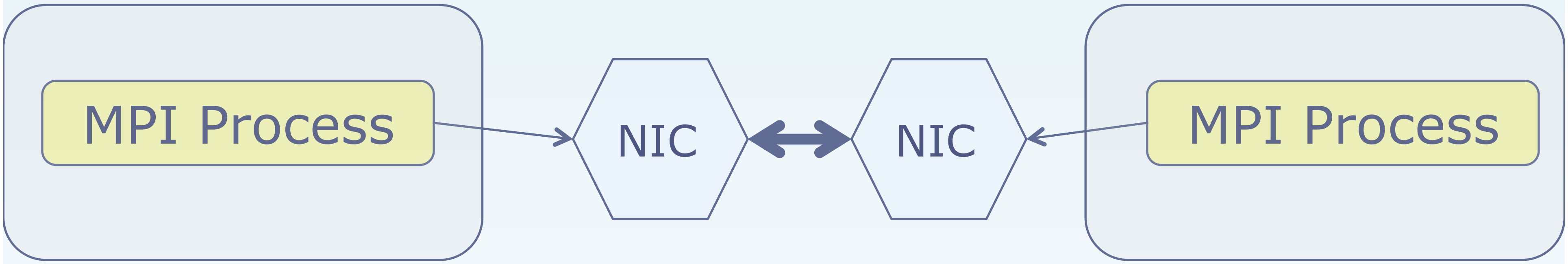
William Gropp
wgropp.cs.illinois.edu

National Center for Supercomputing Applications
University of Illinois at Urbana–Champaign

What do applications need?

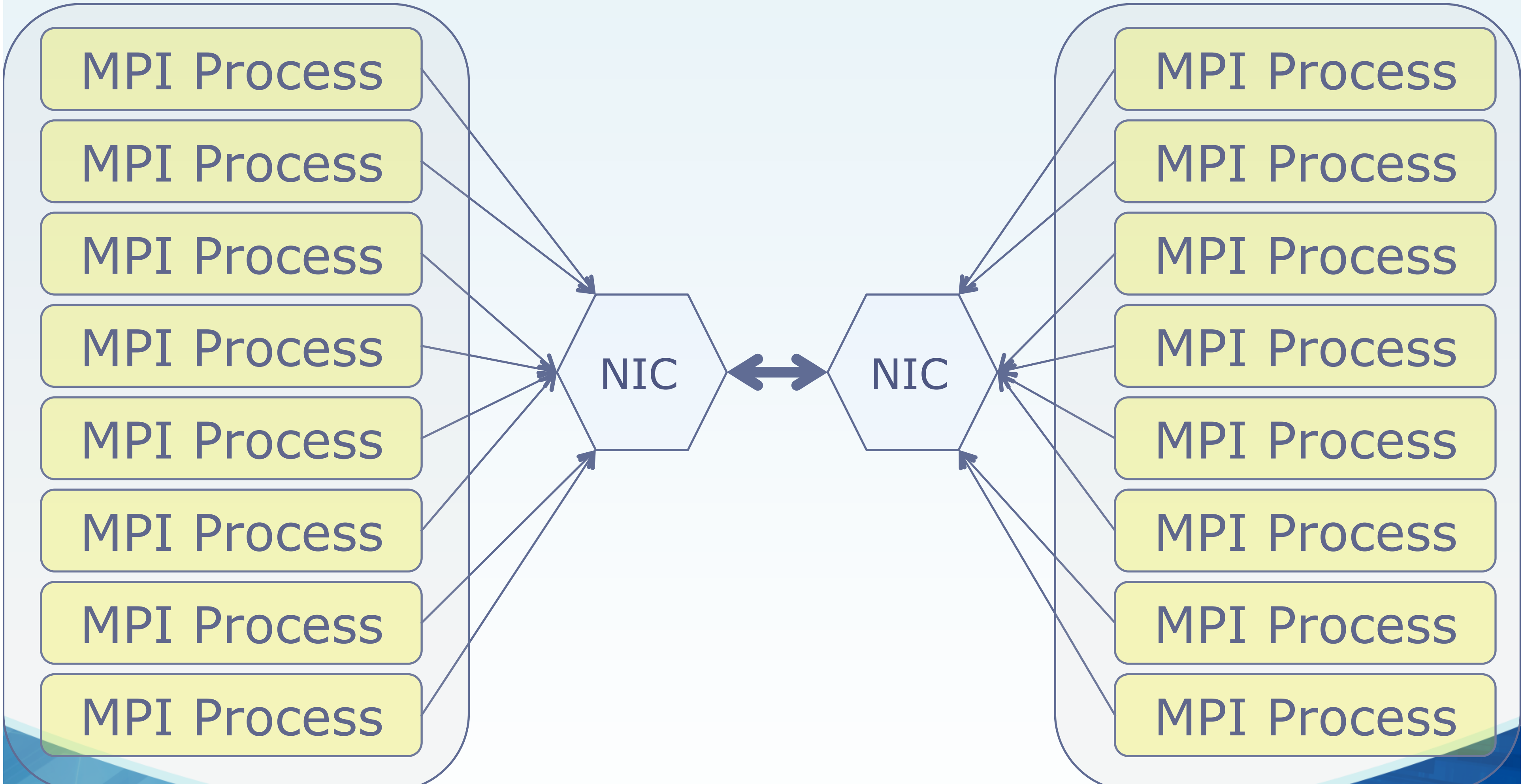
- How do most applications developers view the system?
  - This impact how they write their programs
- What programming approaches might they use?
  - And how do they work together
- How must they model performance
  - And how do communication optimizations impact that
- What features don’t they use but could or should?
  - Often a chicken-and-egg problem

The Most Common Application View



- The MPI everywhere model
- Matches the “postal” performance model T = s + r n
- Variations include multi-threaded processes

A Better Model: MPI Everywhere on SMPs



Reality: Likely Exascale Architectures

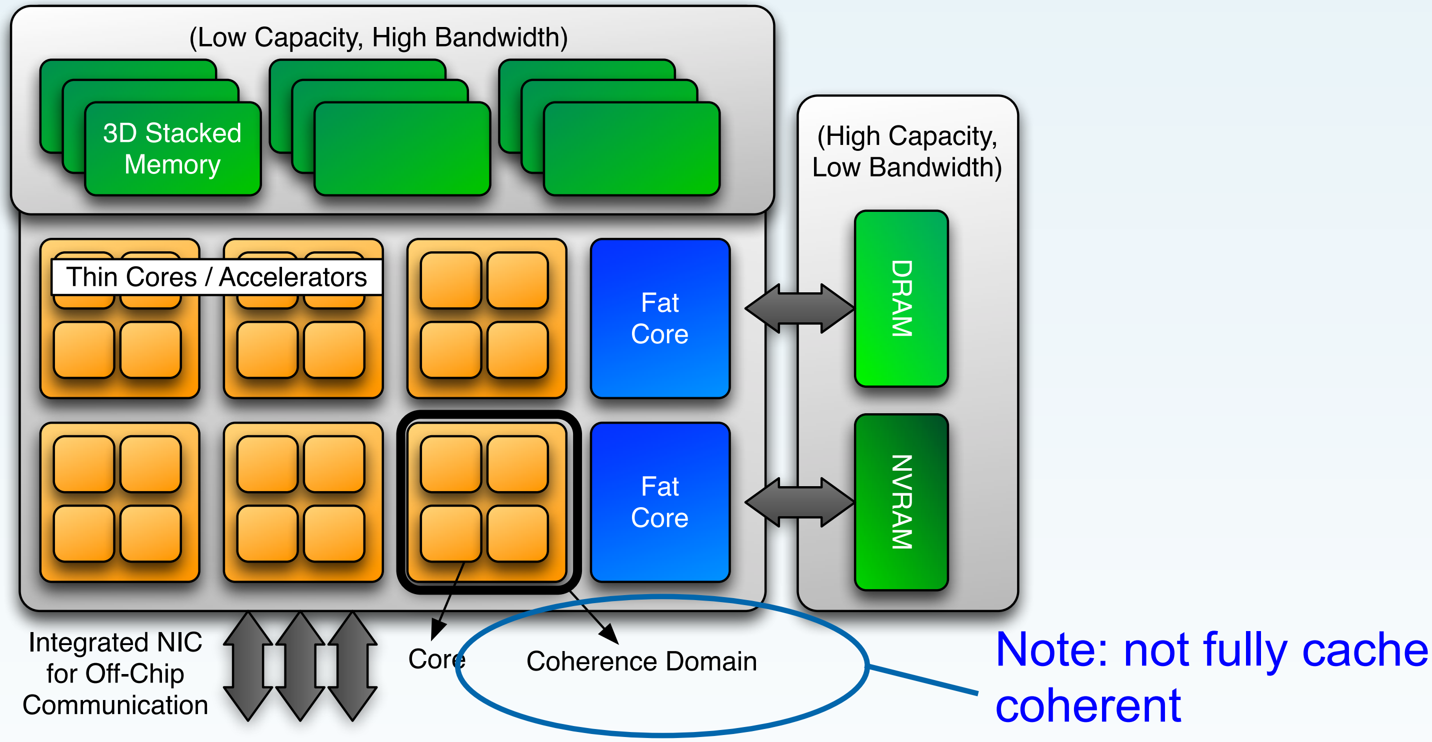
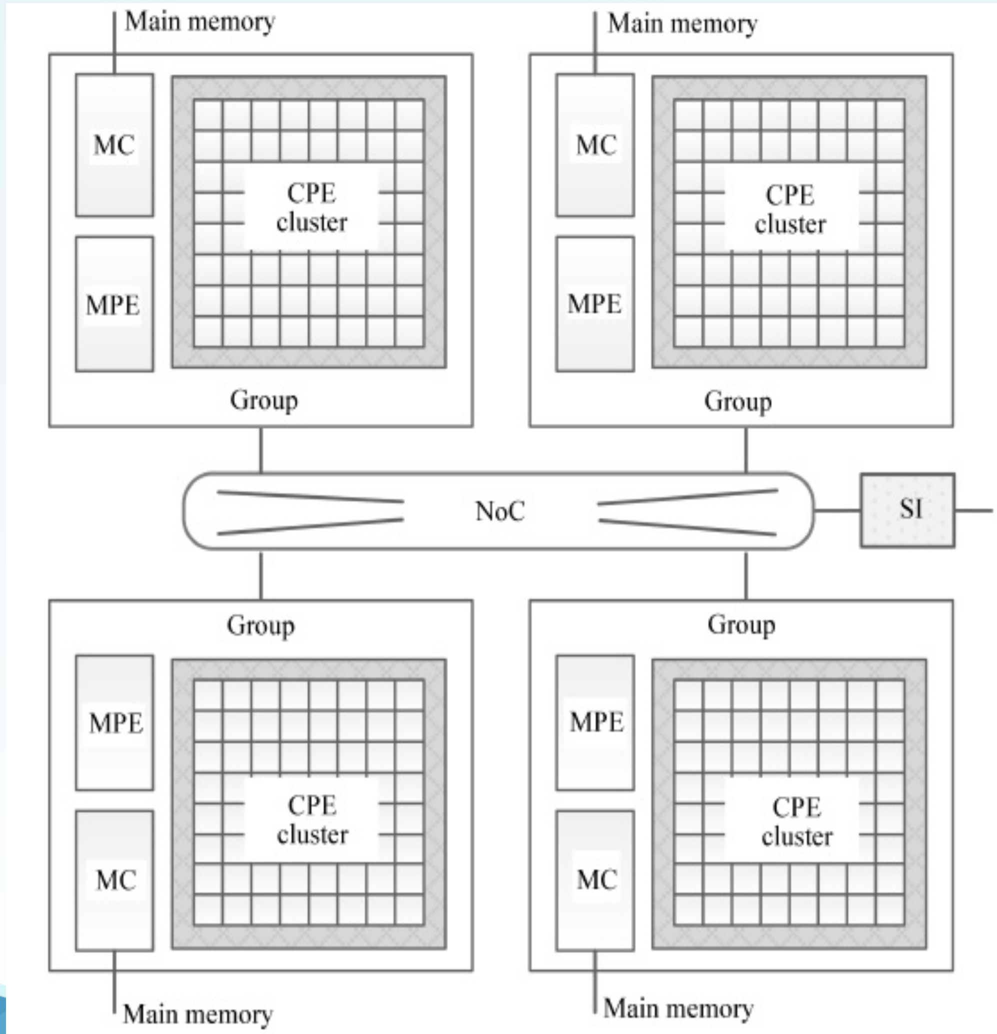


Figure 2.1: Abstract Machine Model of an exascale Node Architecture

- From "Abstract Machine Models and Proxy Architectures for Exascale Computing Rev 1.1," J Ang et al

Another Pre-Exascale Architecture



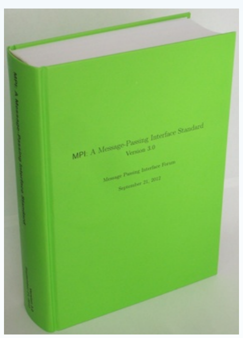
Sunway TaihuLight
- Heterogeneous processors (MPE, CPE)
- No data cache

Programming Models and Systems

- Programming Model: an abstraction of a way to write a program
  - Many levels
    - Procedural or imperative?
    - Single address space with threads?
    - Vectors as basic units of programming?
  - Programming model often expressed with pseudo code
- Programming System: (My terminology)
  - An API that implements parts or all of one or more programming models, enabling the precise specification of a program

Why the Distinction?

- In parallel computing,
    - Message passing is a programming model
        - Abstraction: A program consists of processes that communication by sending messages. See "Communicating Sequential Processes", CACM 21#8, 1978, by C.A.R. Hoare.
    - The Message Passing Interface (MPI) is a programming system
        - Implements message passing and other parallel programming models, including:
        - Bulk Synchronous Programming
        - One-sided communication
        - Shared-memory (between processes)



CUDA/OpenACC/OpenCL are systems implementing a "GPU Programming Model"

- Execution model involves teams, threads, synchronization primitives, different types of memory and operations

Bandwidth, Latency, And All That

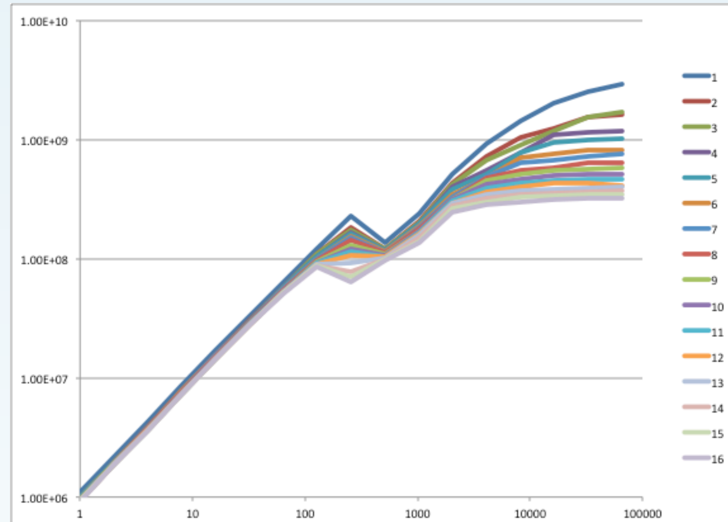
- Bandwidth is easy (and thus gratifying)
  - Asymptotic Bandwidth – its just money
- Latency is more important for productivity and often for performance
- Latency and overhead have many components
  - Propagation delay (because controlled by physics)
    - Quick question: How big is your favorite system measured in clock ticks?
- Which latency and bandwidth terms are important?
  - You mean there are more than one…

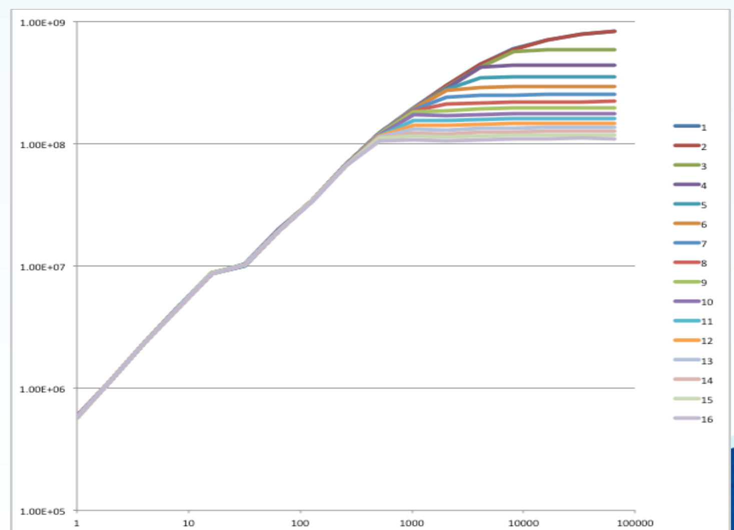
Classic Performance Model

- s + rn
- Model combines overhead and network latency (s) and a single communication rate 1/r
- Good fit to machines when it was introduced
- But does it match modern SMP-based machines?
  - Lets look at the the communication rate per process with processes communicating between two nodes

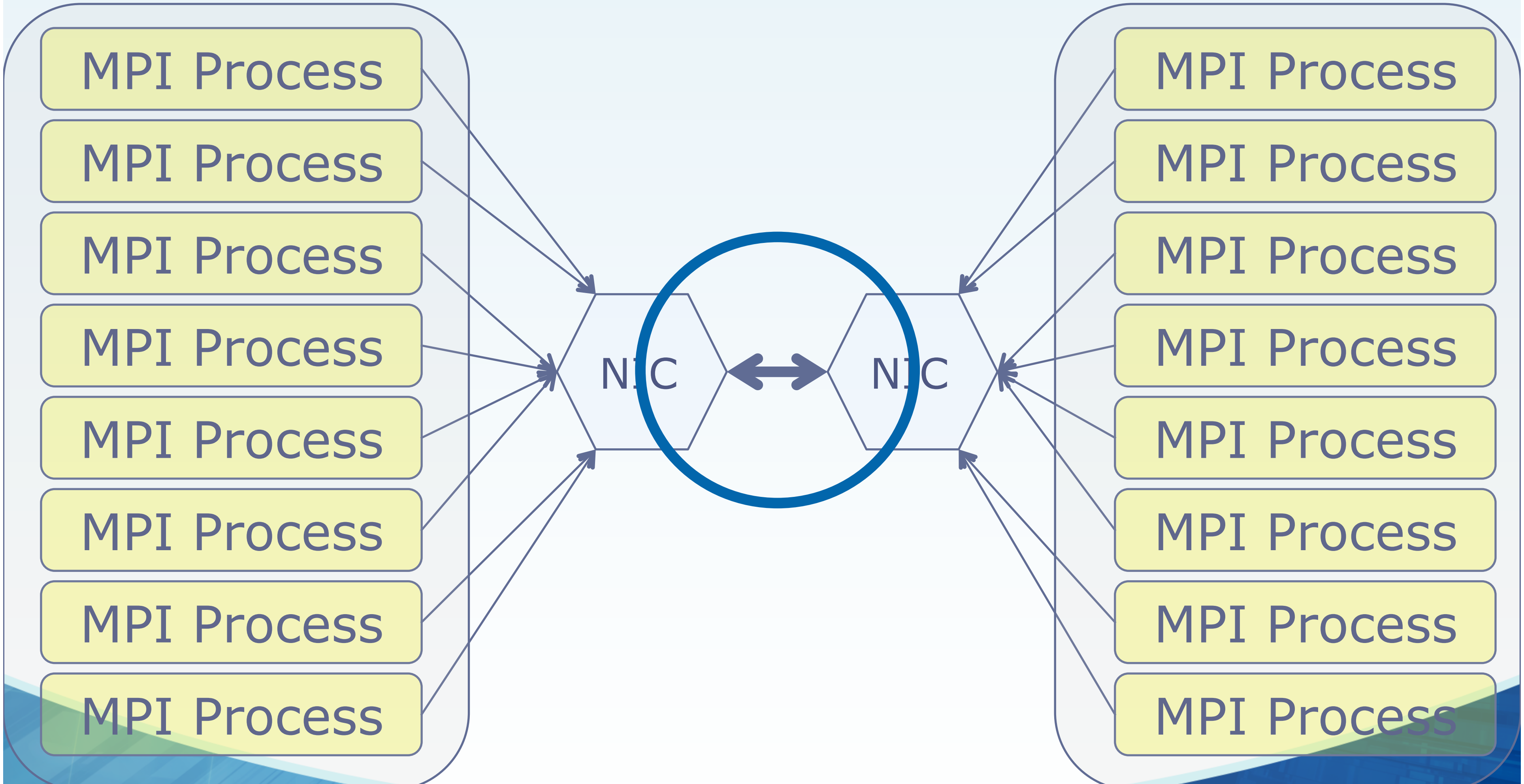
Rate per MPI Process, Node-to-node

- Top is Cray XE6, bottom is IBM Blue Gene/Q
- Rate is measured between 1-k MPI processes on one node, sending to the same number of MPI processes on another node
- If processes did *not* impact each other, there'd be a single curve
- Note short (eager) mostly independent of k





SMP Nodes: One Model



A Slightly Better Model

- Assume that the sustained communication rate is limited by both
  - The maximum rate along any shared link
    - The link between NICs
  - The aggregate rate along parallel links
    - Each of the “links” from an MPI process to/from the NIC

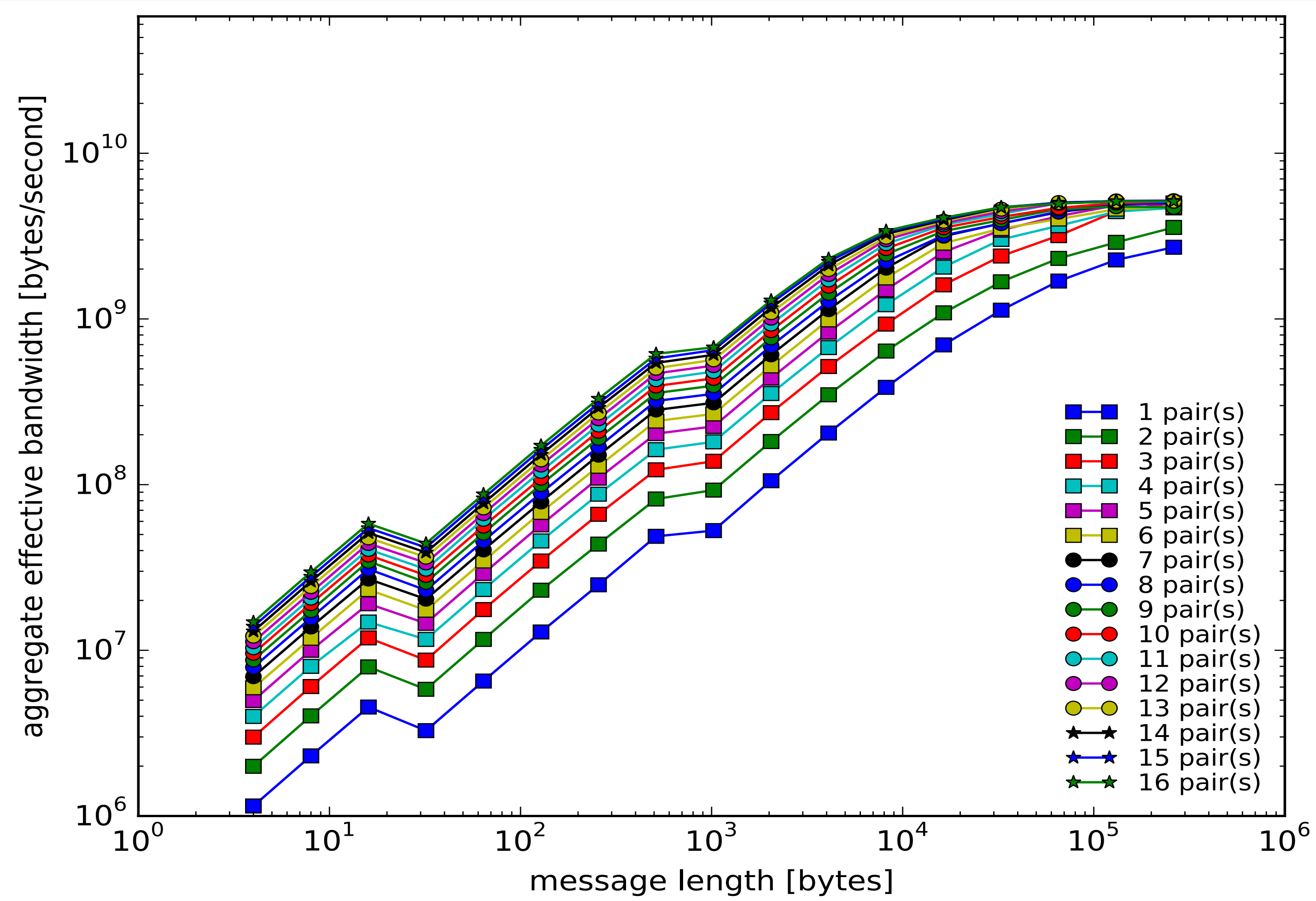
A Slightly Better Model

- For k processes sending messages, the sustained maximum rate is
  - $\min(R_{NIC\text{-}NIC}, k\, R_{CORE\text{-}NIC})$
- Thus
  - $T = s + k\, n/\min(R_{NIC\text{-}NIC}, k\, R_{CORE\text{-}NIC})$
- Note if $R_{NIC\text{-}NIC}$ is very large (very fast network), this reduces to
  - $T = s + k\, n/(k\, R_{CORE\text{-}NIC}) = s + n/R_{CORE\text{-}NIC}$

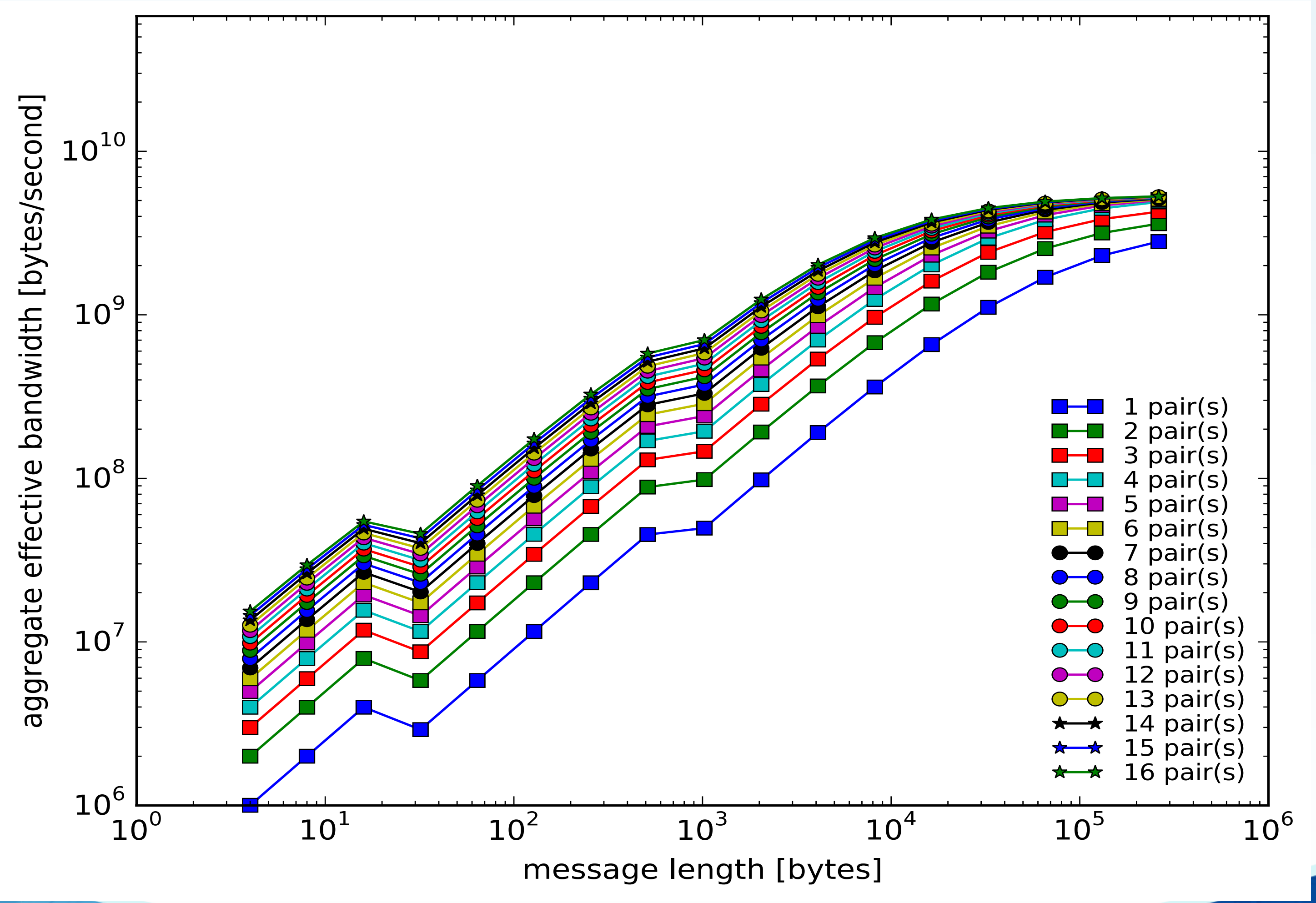
Example: 4 parameter model values for Cray XE6 (Blue Waters)

- 4th parameter uses a different rate for the first process to send and the 2nd etc. processes
  - Does improve fit, but only a little because R_N/R_C is small
- $R_N = R_{NIC}$; $R_C = R_{CORE\text{-}NIC}$
- Short regime
  - s = 4 usec, R_{Cb} = 0.63 GB/s, R_{Ci}=-0.18GB/s, $R_N=\infty$
- Eager regime
  - s = 11 usec, R_{Cb} = 1.7GB/s, R_{Ci} = 0.062GB/s, $R_N=\infty$
- Rendezvous regime
  - s = 20 usec, R_{Cb} = 3.6 GB/s, R_{CI}=0.61GB/s, R_N = 5.5 GB/s

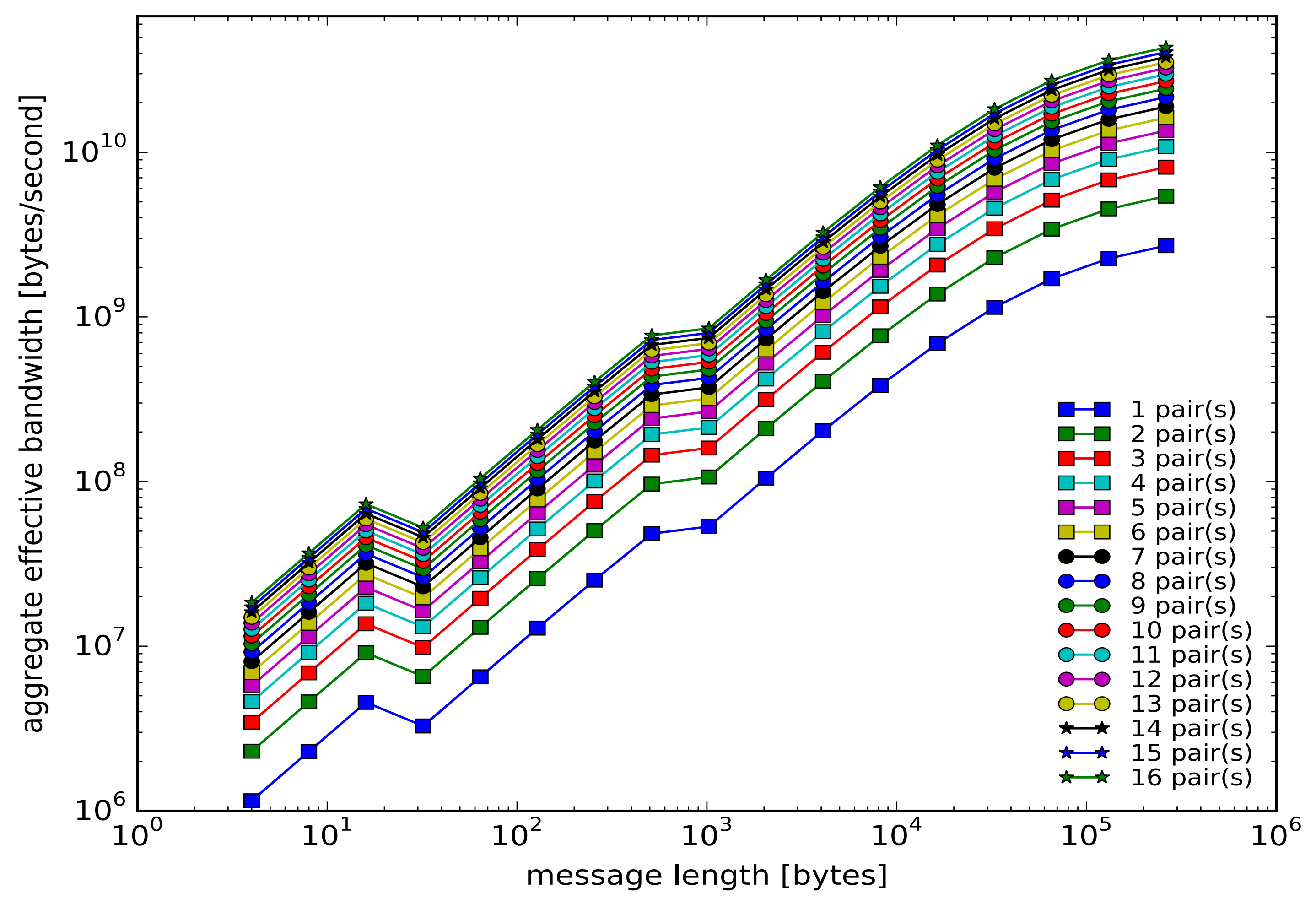
Cray: Measured Data



Cray: 3 parameter model



Cray: 2 parameter model (the standard)

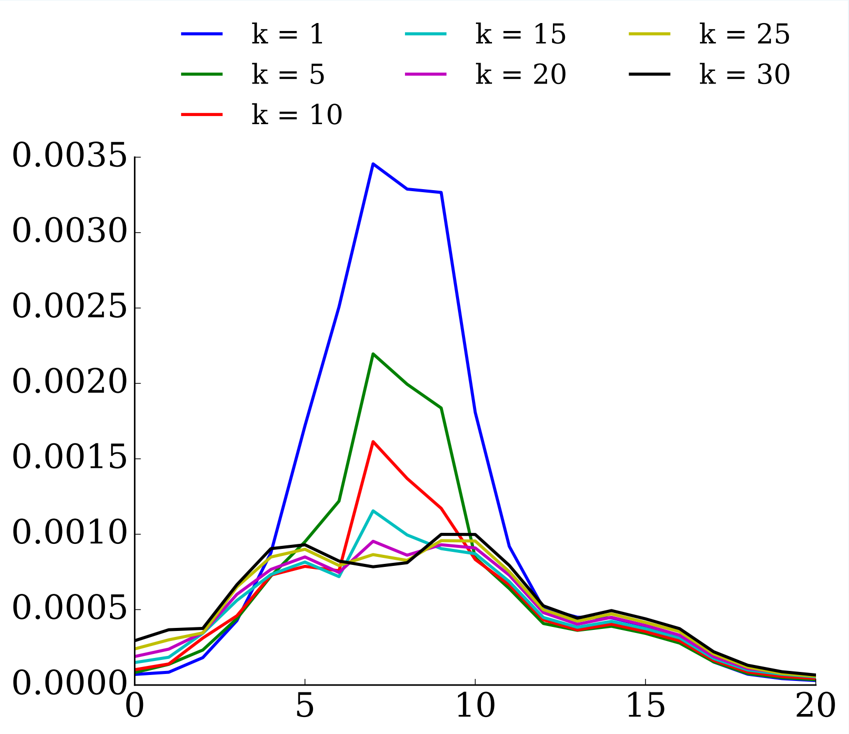


Programming System Overhead

- Overhead (as distinct from latency) comes from many sources
- Examples from MPI:
  - MPI as a library adds library overhead
    - Call overhead
    - Runtime evaluation (e.g, how long is an MPI_INTEGER?)
  - Message-passing adds data to move and interpret
    - Message “envelope”, typically including:
      - Tag, source rank, communicator context, message length, protocol (e.g., eager or rendezvous)
    - How many bits do you use for each?
    - How does that impact message latency? Note message match performance is more than just tag matching

Example: Message Matching in Real Applications

- Case: Messages for multigrid coarse grid exchange
- 1/k of messages sent/received at a time – k=1 is “natural” case
- You can model this (quadratic queue search) but unnatural for application developer
- Yes, can use RMA, but for irregular mesh/matrix, computation of target requires care
- Thanks to Amanda Bienz and Luke Olson for the data



Remote Direct Memory Access and Update

- MPI defines a rich set of read-modify-write operations, including a lower-runtime overhead (read: simpler calling sequence) version (Get_accumulate vs. Fetch_and_op)
  - What happens when the same location is the target of different operations?
  - What is the atomicity of updates? Element? Block? CacheLine?
- The programming system requires all combinations to interact correctly
  - If not, may have to *always* fall back to software (!! :( )
  - MPI is willing to make **informed** restrictions to enable performance if there is modest impact on generality
  - Help us!

Collective Communication and Scalability

- Some of the most efficient algorithms for solving large systems of equations make use of an Allreduce operation
  - These are Krylov algorithms, including conjugate gradient and GMRES
  - Yes, there are alternate algorithms, but usually have worse time-to-solution; there are sound mathematical reasons for this
- The following analysis is from Paul Fischer, taken from his Nek5000 CFD code
  - Demonstrated scalability to over 100k processes – but with the right communications support
  - Analysis based on communication time < computation time
    - Can make true by making problem big enough
    - But science problems usually not arbitrarily large

Scaling Estimates: Conjugate Gradients

$$\frac{T_c}{T_a} = \frac{6\left(1 + \frac{1}{m_2}(n/P)^{2/3} + 4\log_2 P\right)\alpha}{27\,n/P} \lesssim 1$$

$$P = 10^6, \qquad \log_2 P = 20, \qquad \boxed{(n/P) \approx 8500}$$

$$P = 10^9, \qquad \log_2 P = 30, \qquad \boxed{(n/P) \approx 12000}$$

- The inner-products in CG, which give it its optimality, drive up the minimal effective granularity because of the log P scaling of all_reduce.

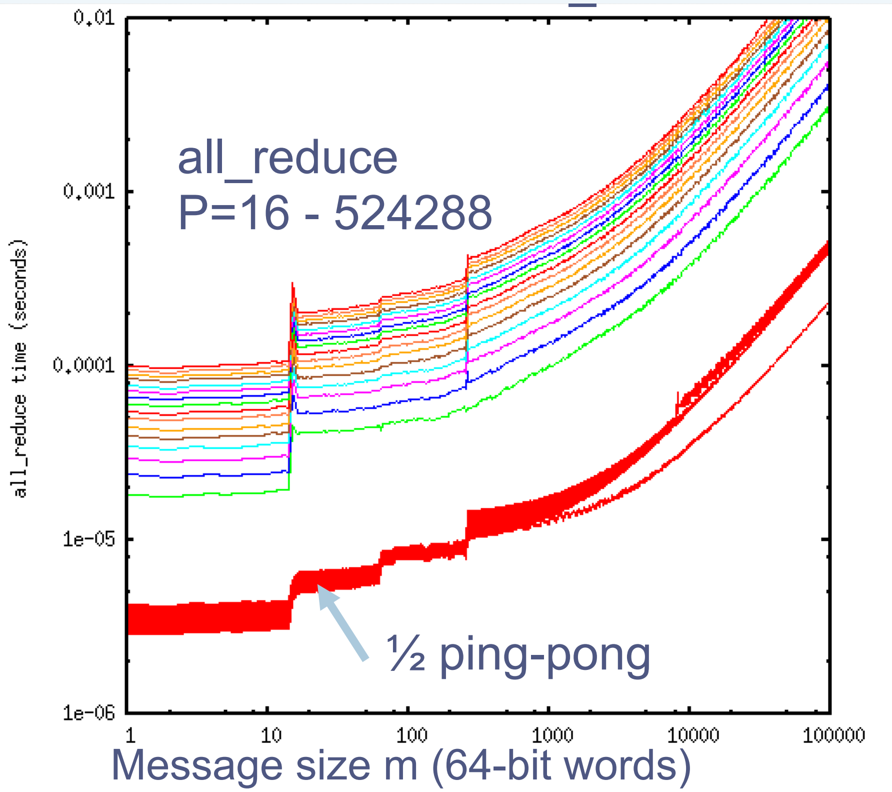
- On BG/L, /P, /Q, however, all_reduce is effectively P-independent.

Thanks to Paul Fischer

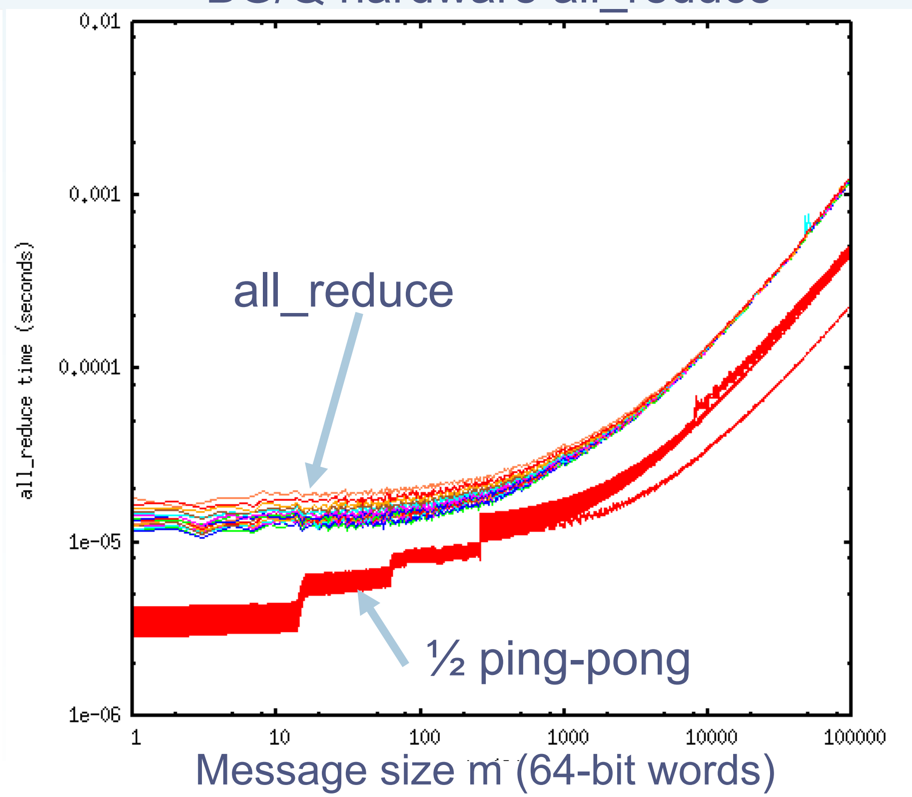
Eliminating log P term in CG

- On BG/L, /P, /Q, all_reduce is nearly ***P-independent***.
- For P=524288, all_reduce(1) is only 4α !

BG/Q software all_reduce



BG/Q hardware all_reduce



Thanks to Paul Fischer

Eliminating log P term in CG

2 x 4

$$\frac{T_c}{T_a} = \frac{6\left(1 + \frac{1}{m_2}(n/P)^{2/3} + 4\log_2 P\right)\alpha}{27\,n/P} \leq 1$$

$$\boxed{n/P \approx 1200}$$

- On BG/L, /P, /Q, CG is effectively P-independent because of *hardware supported all_reduce*.
- In this (admittedly simple) exascale model, net result is a 10x improvement in granularity (n/P=1200 vs. 12,000).

→ 10x faster run, but no reduction in power consumption.

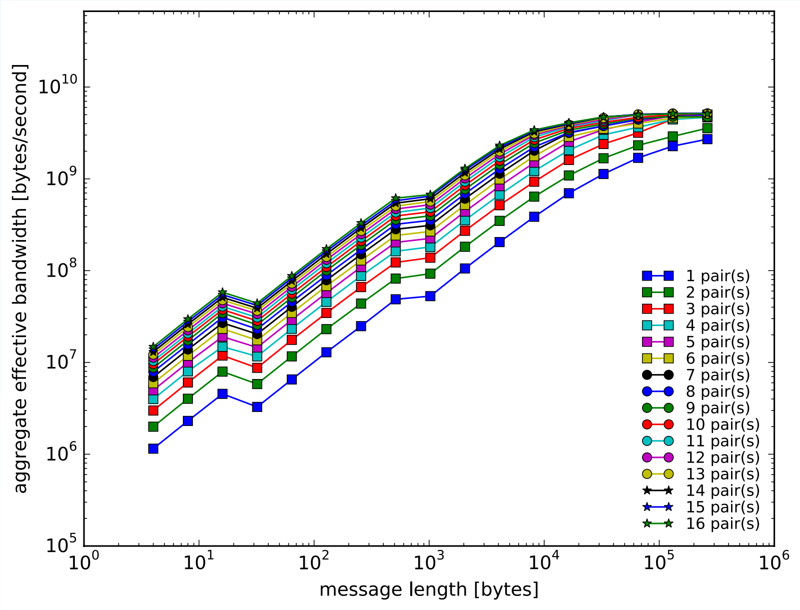
Thanks to Paul Fischer

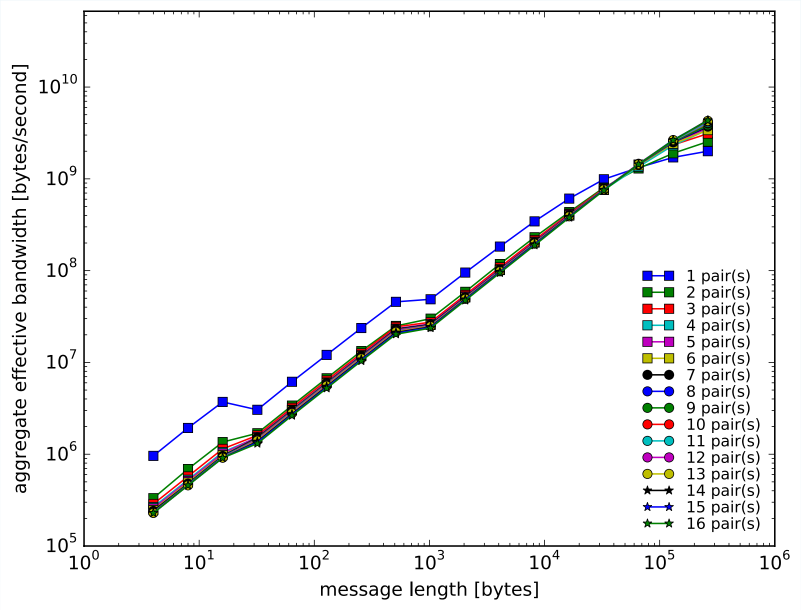
NCSA

The overhead of the “+” in MPI + X

- How do you combine different communications paths (e.g., network + shared memory)?
- Functionality isn’t enough – what is the performance cost?
  - Often the only correct solution is to poll
  - Note issue with Active Message work – many results used either poll (fast) or interrupt (responsive)
- Thread-safety
  - Do you need memory barriers? Critical sections?
  - How do you handle the issues described in “Threads Cannot be Implemented as a Library”?
    - Without forcing pthread lock/unlock everywhere (ask me how I know :)) ?
  - Many (but not all) current systems struggle to give good performance

Results for Multithreaded Ping Pong Benchmark
Coarse-Grained Locking

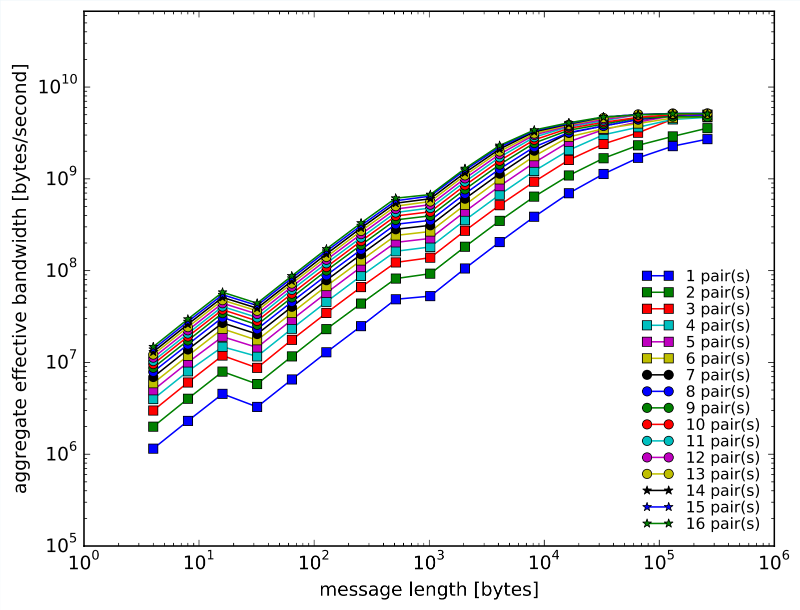


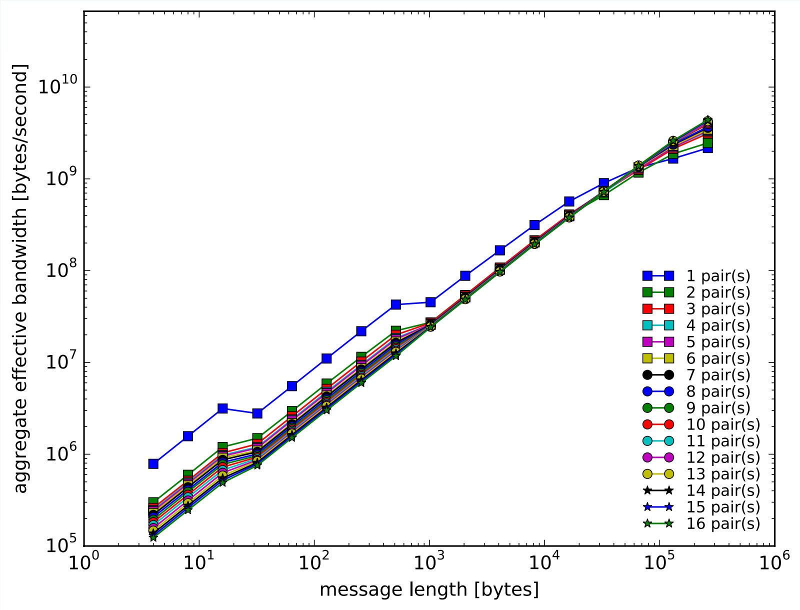


Measurements for single-threaded benchmark

Measurements for multi-threaded benchmark

Results for Multithreaded Ping Pong Benchmark
Fine-Grained Locking





Measurements for single-threaded benchmark

Measurements for multi-threaded benchmark

Overlap of Communication and Computation

- Example: “Halo Exchange”
  - Send surface of a data cube to neighbor processes
  - By now, have trained MPI programmers to use
    - Do (all neighbors) MPI_Isend(…)
      Do (all neighbors) MPI_Irecv(…)
      MPI_Waitall(…)
- But this is no longer sufficient for acceptable performance in most cases…

Halo Exchange on BG/P and Cray XT4

- 2048 doubles to each neighbor
- Rate is MB/Sec (for all tables)

BG/P	4 Neighbors		8 Neighbors	
	Irecv/Send	Irecv/Isend	Irecv/Send	Irecv/Isend
World	208	328	184	237
Even/Odd	219	327	172	243
Cart_create	301	581	242	410

Cray XT4	4 Neighbors			8 Neighbors	
	Irecv/Send	Irecv/Isend	Phased	Irecv/Send	Irecv/Isend
World	311	306	331	262	269
Even/Odd	257	247	279	212	206
Cart_create	265	275	266	236	232

Halo Exchange on BG/Q and Cray XE6

- 2048 doubles to each neighbor
- Rate is MB/sec (for all tables)

BG/Q	**8 Neighbors**	
	Irecv/Send	Irecv/Isend
World	662	1167
Even/Odd	711	1452
1 sender		2873

Cray XE6	**8 Neighbors**	
	Irecv/Send	Irecv/Isend
World	352	348
Even/Odd	338	324
1 sender		5507

How Fast "should" it be?

- Lets look at a single process sending to its neighbors.
- Based on our performance model, we *expect* the rate to be roughly twice that for the halo (since this test is only sending, not sending and receiving)

System	4 neighbors		8 Neighbors	
		Periodic		Periodic
BG/L	488	490	389	389
BG/L, VN	294	294	239	239
BG/P	1139	1136	892	892
BG/P, VN	468	468	600	601
XT3	1005	1007	1053	1045
XT4	1634	1620	1773	1770
XT4 SN	1701	1701	1811	1808

Comparing Rates

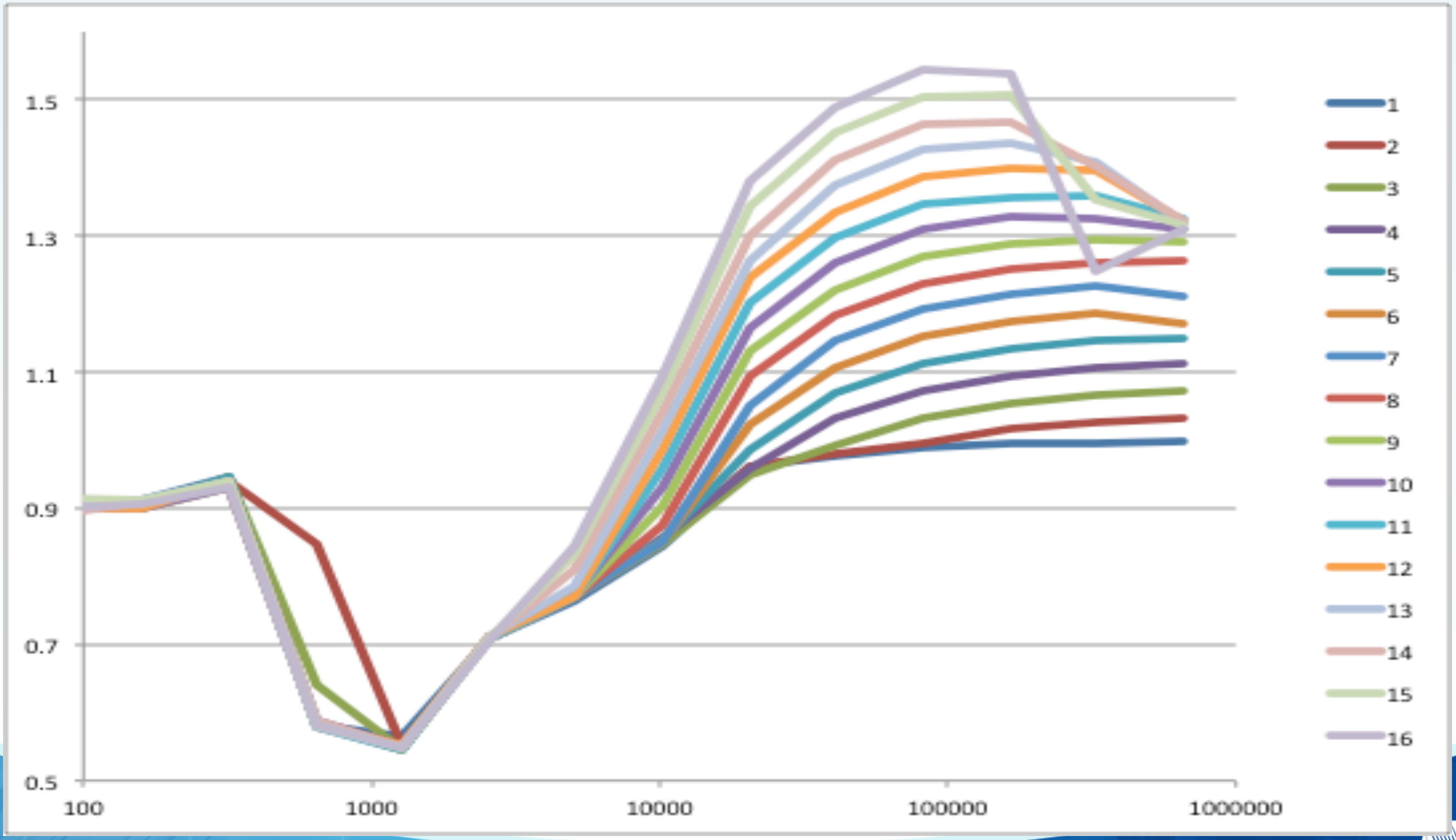
- Ratios of a single sender to all processes sending (in rate)
- *Expect* a factor of roughly 2 (since processes must also receive)

System	4 neighbors		8 Neighbors	
		Periodic		Periodic
BG/L	2.24		2.01	
BG/P	3.8		2.2	
BG/Q			1.98	
XT3	7.5	8.1	9.08	9.41
XT4	10.7	10.7	13.0	13.7
XE6			15.6	15.9

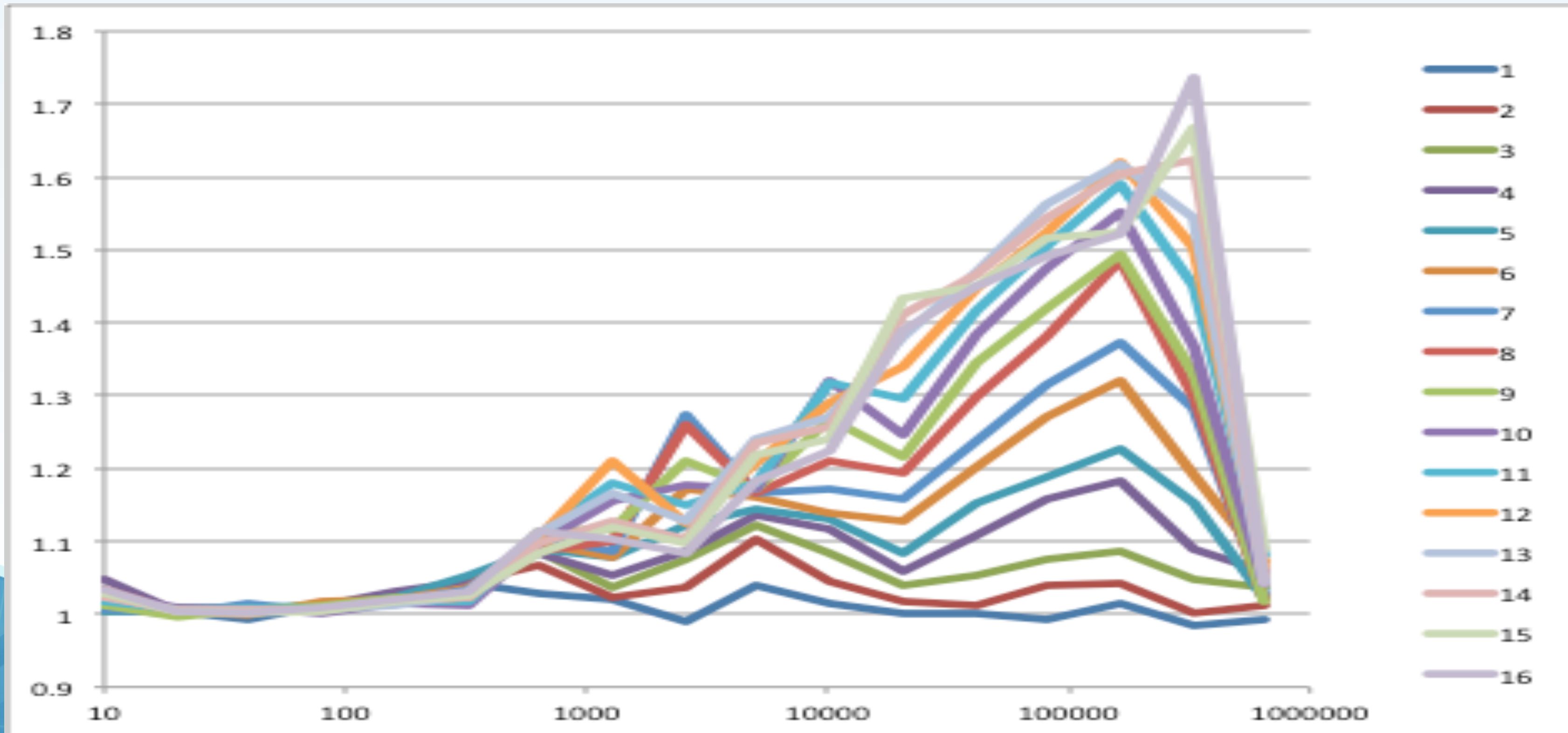
- BG gives roughly double the halo rate. XTn and XE6 are much higher.
- Explanation: $R_N << k R_C$ on Cray

Does Communication Overlap Help? (BG/Q)

- Graph show performance advantage to using overlap as a function of work size (message size = 1/10 work)

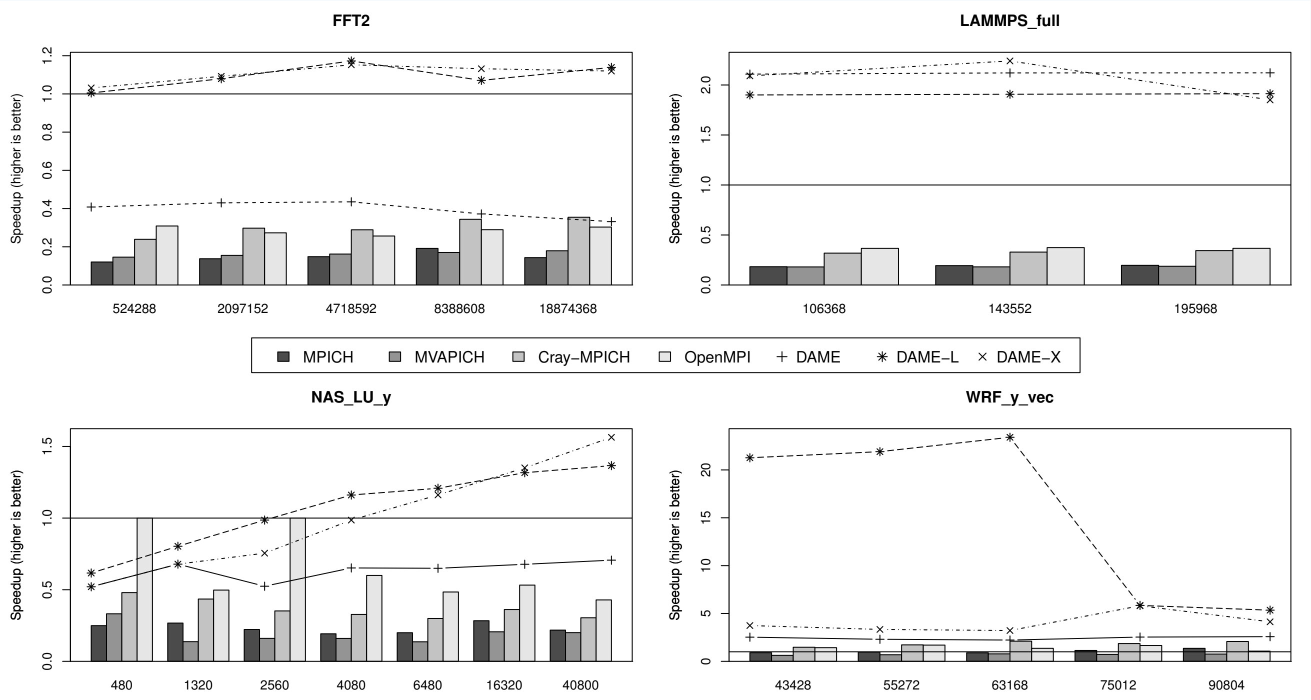


Does Communication Overlap Help? (Cray XE6)



Data to be moved is not always contiguous

- MPI datatypes provide a way to compactly represent many data patterns
- High performance is possible with proper care
- MPI_Type_commit provides opportunity to optimize (compile code in our case)



DAME: A Runtime-Compiled Engine for Derived Datatypes, Tarun Prabhu and William Gropp, Proceedings of the 22nd European MPI Users' Group Meeting, 4:1–4:10, 2015

Dynamic Membership

- MPI has a collective model for dynamically changing the number of processes in a parallel job
- MPI's API intended to support scale (add hundreds – thousands of nodes/processes quickly)
  - Unimplemented – why? What needs to be done? Is the MPI API a problem, or is it a chicken and egg problem (no demand because it doesn't work because there is no demand)
- A similar capability is needed for some approaches to fault tolerance
- A related (perhaps) issue is startup efficiency. A parallel job should be able to start in < 1sec even one 100K nodes
  - Time to send code with broadcast algorithm < 1sec
  - On demand connection + implicit info, distributed tables should remove serial bottlenecks
  - Etc. :)

NCSA

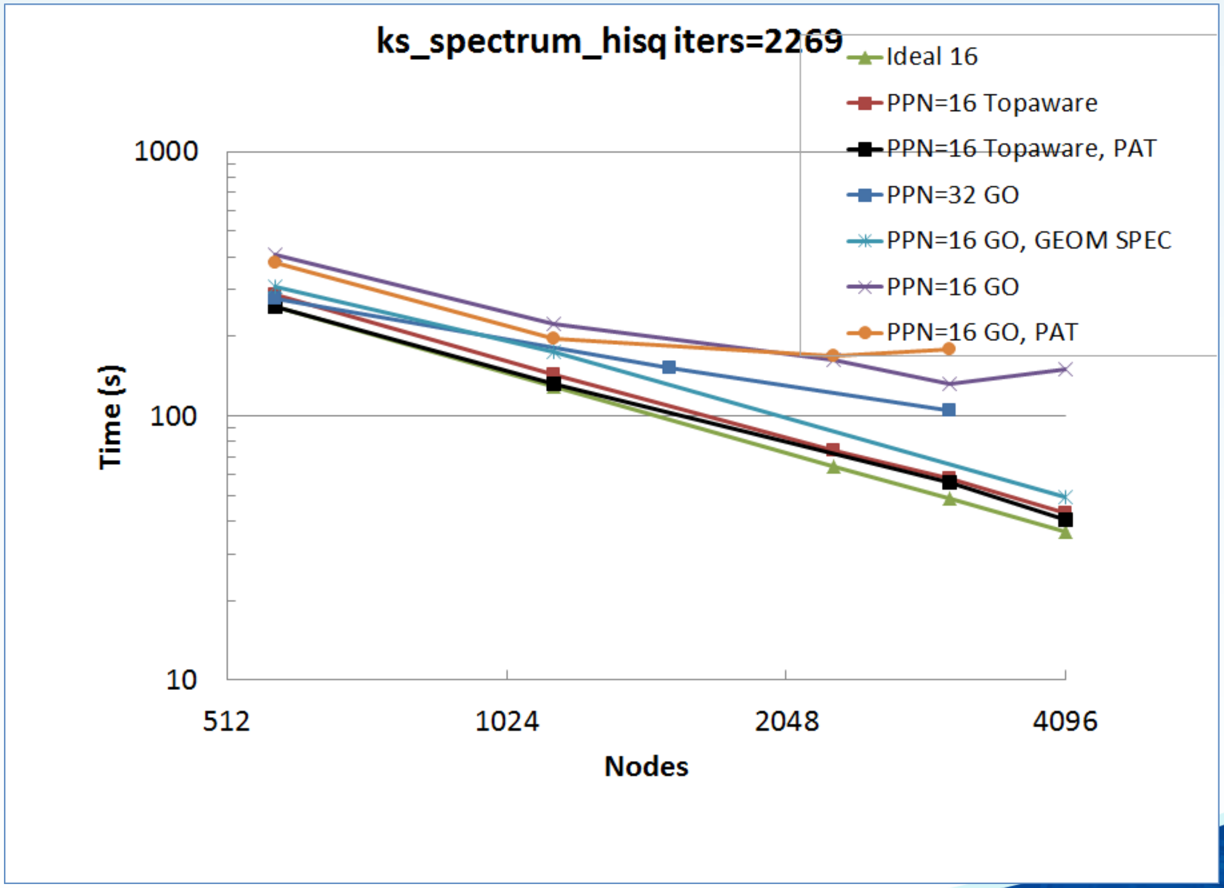
Sharing with Others

- Applications rarely have the entire machine to themselves
- Thus their communication performance may be impacted by other users or the system
  - Other users, if messages must share communication links
  - The system, e.g., for I/O operations including backup
- How should jobs be laid out on a system to provide
  - Good application performance
  - Good system utilization
- Not easy, even with simple interconnect topologies
- Example: Topology-Aware Scheduling for Blue Waters (Cray XE6/XK7; Torus interconnect)
  - Thanks to Jeremy Enos and his team

Scaling effect example (MILC)

1.45x speedup at 576 nodes

Near linear scaling only possible with TAS placement



Summary
What do Applications Want?

- Performance and productivity
  - Low Latency is very important
    - Consider $n_{1/2}$ as a figure of merit
  - Fast key collectives esp. MPI_Allreduce
- Full performance from node
  - Communication/computation overlap, progress
  - Efficient handling of intra-node and inter-node communication at the same time (the “+” in MPI+X)
- Predictable performance
  - Minimal impact from other jobs (may require topology aware scheduling)
- Support for efficient non-contiguous data moves
- Support for fast remote RMW operations