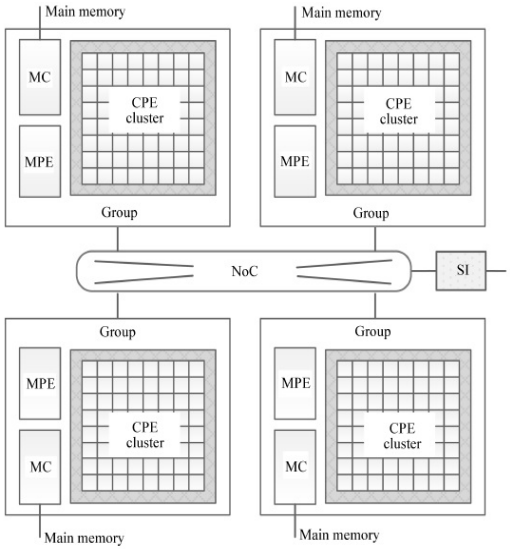
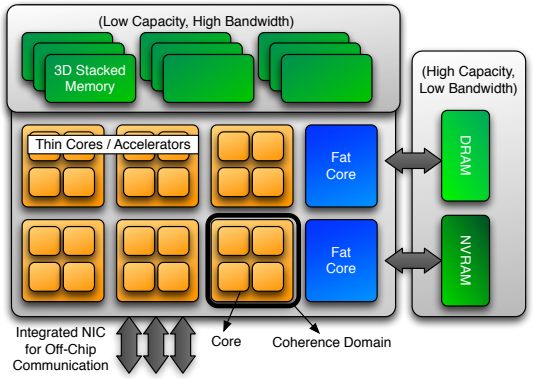
MPI+X for Extreme Scale Computing

William Gropp
http://wgropp.cs.illinois.edu

Some Likely Exascale Architectures





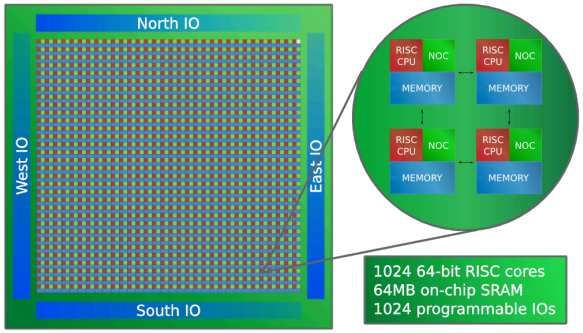


Figure 2.1: Abstract Machine Model of an exascale Node Architecture

Sunway TaihuLight

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

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

Adapteva Epiphany-V

- 1024 RISC processors
- 32x32 mesh
- Very high power efficiency

MPI (The Standard) Can Scale Beyond Exascale

- MPI implementations already supporting more than 1M processes
  - Several systems (including Blue Waters) with over 0.5M independent cores
- Many Exascale designs have a similar number of nodes as today's systems
  - MPI as the internode programming system seems likely
- There are challenges
  - Connection management
  - Buffer management
  - Memory footprint
  - Fast collective operations
  - …
  - And no implementation is as good as it needs to be, but
  - There are no intractable problems here – MPI implementations can be engineered to support Exascale systems, even in the MPI-everywhere

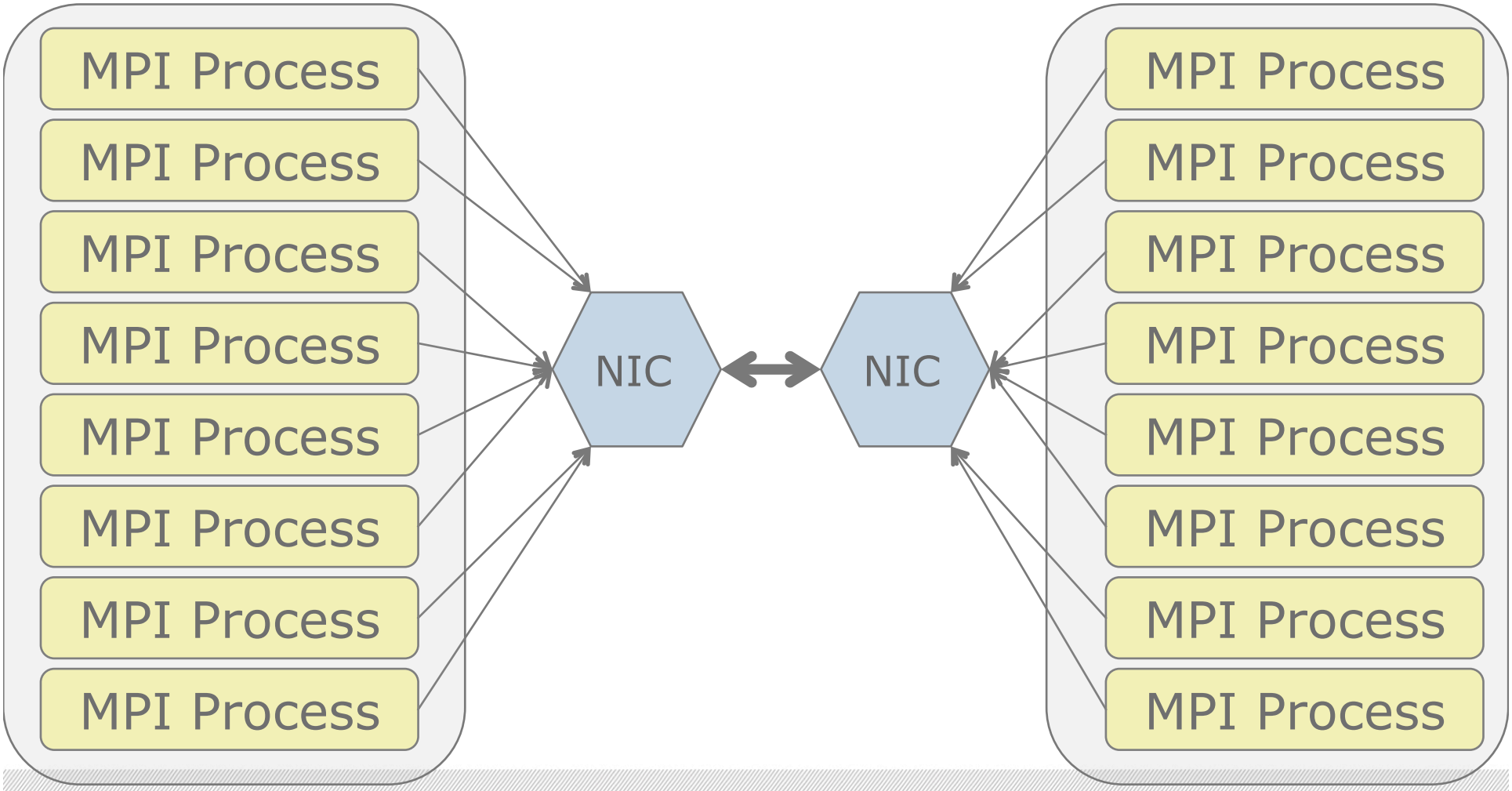
Applications Still Mostly MPI-Everywhere

- "the larger jobs (> 4096 nodes) mostly use message passing with no threading." – BW Workload study, https://arxiv.org/ftp/arxiv/papers/1703/1703.00924.pdf
- Benefit of programmer-managed locality
  - Memory performance nearly stagnant
  - Parallelism for performance implies locality must be managed effectively
- Benefit of a single programming system
  - Often stated as desirable but with little evidence
  - Common to mix Fortran, C, Python, etc.
  - But…Interface between systems must work well, and often don't
    - E.g., for MPI+OpenMP, who manages the cores and how is that negotiated?

Why Do Anything Else?

- Performance
  - May avoid memory (though usually not cache) copies
- Easier load balance
  - Shift work among cores with shared memory
- More efficient fine-grain algorithms
  - Load/store rather than routine calls
  - Option for algorithms that include races (asynchronous iteration, ILU approximations)
- Adapt to modern node architecture…

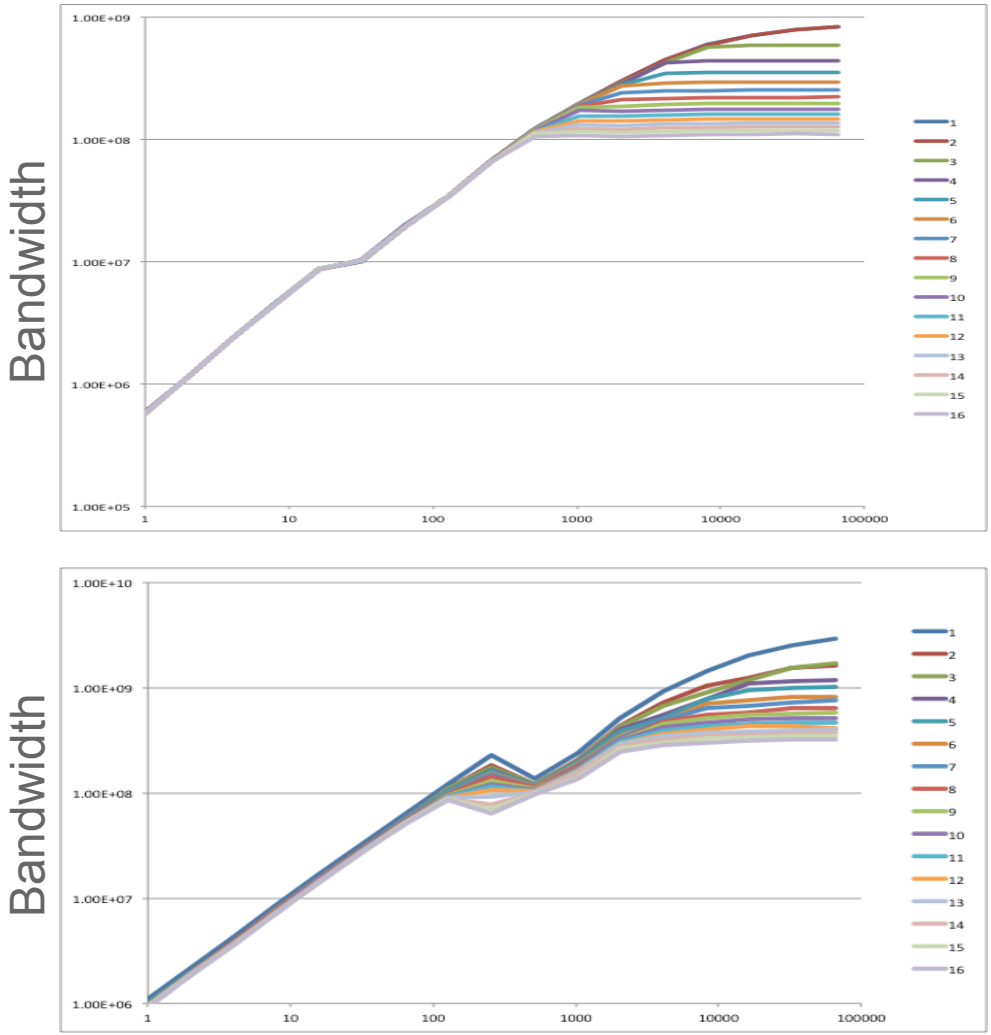
SMP Nodes: One Model



Classic Performance Model

- s + r n
  - Sometimes called the “postal model”
- Model combines overhead and network latency (s) and a single communication rate 1/r for n bytes of data
- Good fit to machines when it was introduced
- But does it match modern SMP-based machines?
  - Let’s look at the the communication rate per process with processes communicating between two nodes

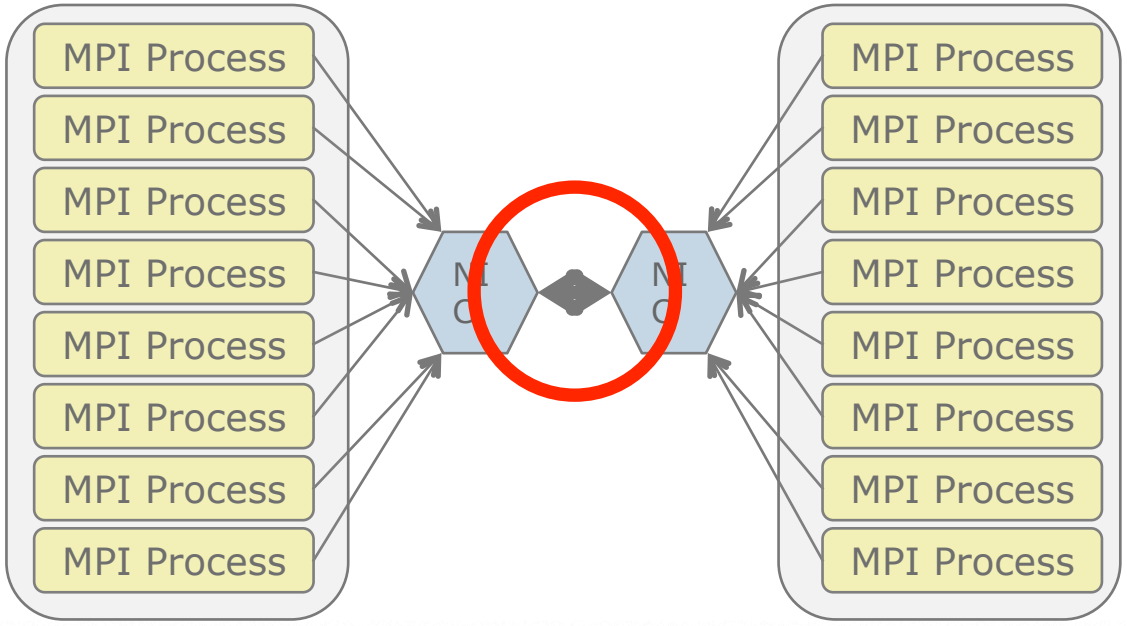
Rates Per MPI Process



- Ping-pong between 2 nodes using 1-16 cores on each node
- Top is BG/Q, bottom Cray XE6
- "Classic" model predicts a single curve – rates independent of the number of communicating processes

Why this Behavior?

- The T = s + r n model predicts the *same* performance independent of the number of communicating processes
  - What is going on?
  - How should we model the time for communication?



Modeling the Communication

- Each link can support a rate r_L of data
- Data is pipelined (Logp model)
  - Store and forward analysis is different
- Overhead is completely parallel
  - k processes sending one short message each takes the same time as one process sending one short message

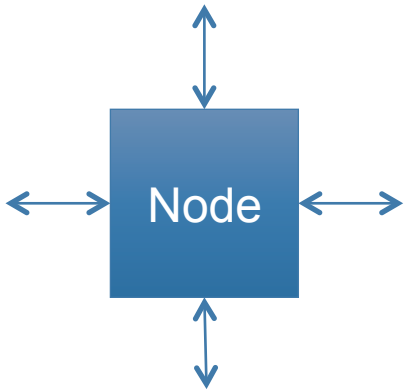
A Slightly Better Model

- For k processes sending messages, the sustained 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
  - $T = s + k\ n/(k\ R_{CORE\text{-}NIC}) = s + n/R_{CORE\text{-}NIC}$

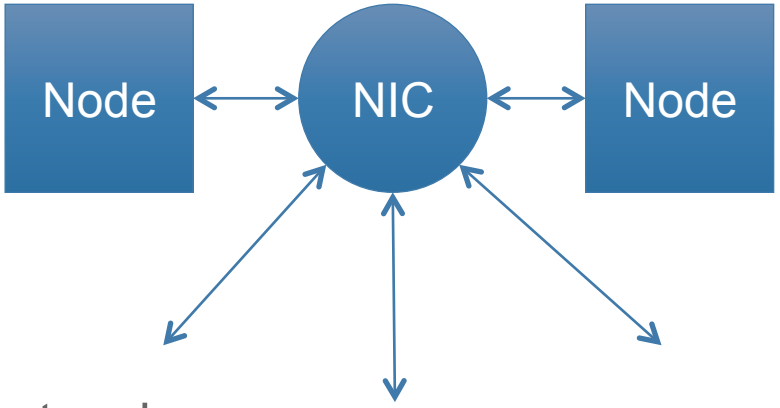
Two Examples

- Two simplified examples:

Blue Gene/Q



Cray XE6



- Note differences:
  - BG/Q : Multiple paths into the network
  - Cray XE6: Single path to NIC (shared by 2 nodes)
  - Multiple processes on a node sending can exceed the available bandwidth of the single path

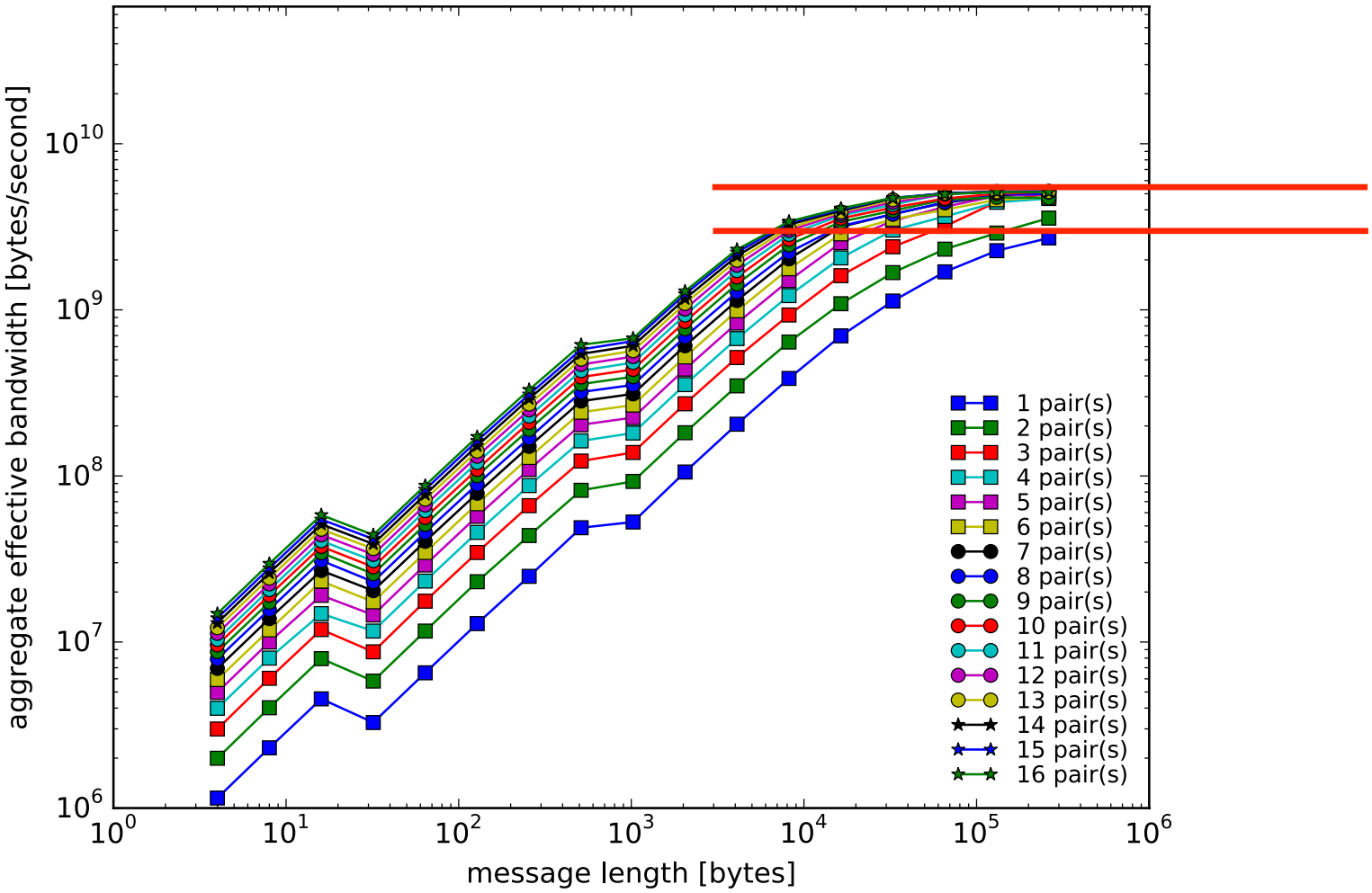
The Test

- Nodecomm discovers the underlying physical topology
- Performs point-to-point communication (ping-pong) using 1 to # cores per node to another node (or another chip if a node has multiple chips)
- Outputs communication time for 1 to # cores along a single channel
  - Note that hardware may route some communication along a longer path to avoid contention.
- The following results use the code available soon at
  - https://bitbucket.org/william gropp/baseenv

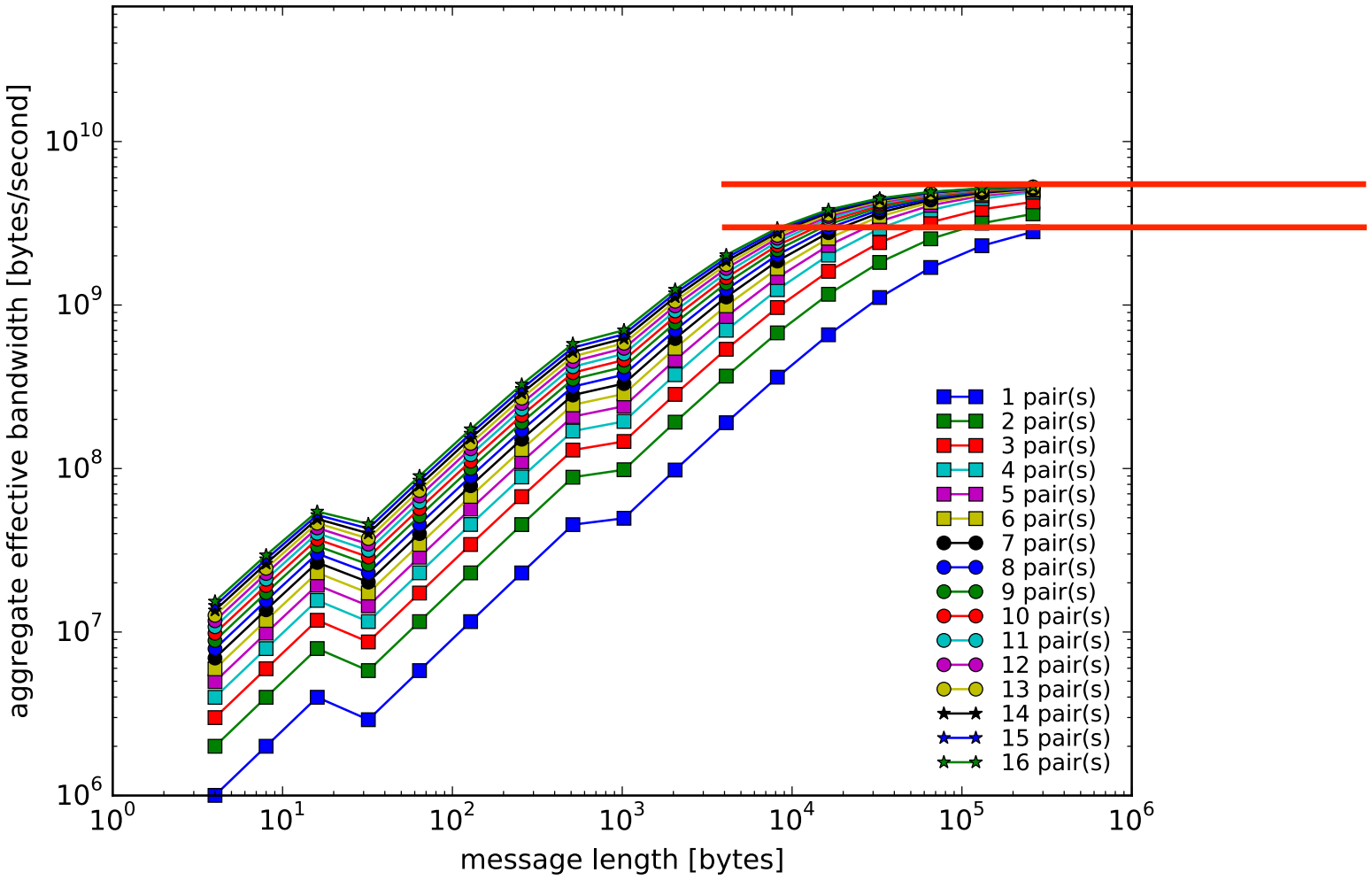
How Well Does this Model Work?

- Tested on a wide range of systems:
  - Cray XE6 with Gemini network
  - IBM BG/Q
  - Cluster with InfiniBand
  - Cluster with another network
- Results in
  - Modeling MPI Communication Performance on SMP Nodes: Is it Time to Retire the Ping Pong Test
    - W Gropp, L Olson, P Samfass
    - Proceedings of EuroMPI 16
    - https://doi.org/10.1145/2966884.2966919
- Cray XE6 results follow

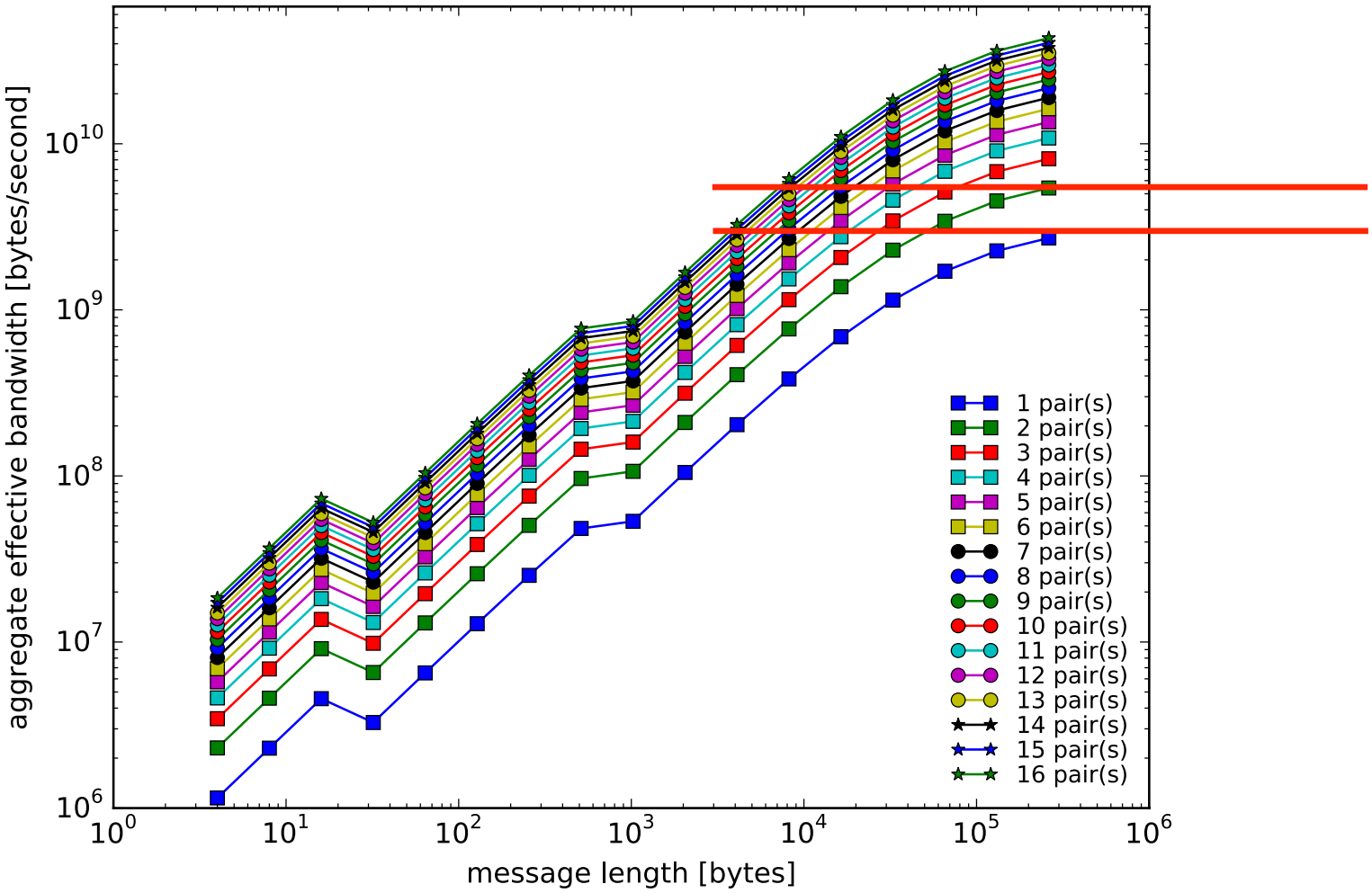
Cray: Measured Data



Cray: 3 parameter (new) model



Cray: 2 parameter model



Notes

- Both Cray XE6 and IBM BG/Q have inadequate bandwidth to support each core sending data along the same link
  - But BG/Q has more independent links, so it is able to sustain a higher effective “halo exchange”

Ensuring Application Performance and Scalability

- Defer synchronization and overlap communication and computation
  - Need to support asynchronous progress
  - Avoid busy-wait/polling
- Reduce off-node communication
  - Careful mapping of processes/threads to nodes/cores
- Reduce intranode message copies…

What To Use as X in MPI + X?

- Threads and Tasks
  - OpenMP, pthreads, TBB, OmpSs, StarPU, …
- Streams (esp for accelerators)
  - OpenCL, OpenACC, CUDA, …
- Alternative distributed memory system
  - UPC, CAF, Global Arrays, GASPI/GPI
- MPI shared memory

X = MPI (or X = ϕ)

- MPI 3.1 features esp. important for Exascale
  - Generalize collectives to encourage post BSP (Bulk Synchronous Programming) approach:
    - Nonblocking collectives
    - Neighbor – including nonblocking – collectives
  - Enhanced one-sided
    - Precisely specified (see “Remote Memory Access Programming in MPI-3,” Hoefler et at, in ACM TOPC)
    - http://dl.acm.org/citation.cfm?doid=2780584
    - Many more operations including RMW
  - Enhanced thread safety

X = Programming with Threads

- Many choices, different user targets and performance goals
  - Libraries: Pthreads, TBB
  - Languages: OpenMP 4, C11/C++11
- C11 provides an adequate (and thus complex) memory model to write portable thread code
  - Also needed for MPI-3 shared memory; see “Threads cannot be implemented as a library”, http://www.hpl.hp.com/techreports/2004/HPL-2004-209.html

What are the Issues?

- Isn't the beauty of MPI + X that MPI and X can be learned (by users) and implemented (by developers) independently?
  - Yes (sort of) for users
  - No for developers
- MPI and X must either partition or share resources
  - User must not blindly oversubscribe
  - Developers must negotiate

More Effort needed on the “+”

- MPI+X won’t be enough for Exascale if the work for “+” is not done very well
  - Some of this may be language specification:
    - User-provided guidance on resource allocation, e.g., MPI_Info hints; thread-based endpoints
  - Some is developer-level standardization
    - A simple example is the MPI ABI specification – users should ignore but benefit from developers supporting

Some Resources to Negotiate

- CPU resources
  - Threads and contexts
  - Cores (incl placement)
  - Cache
- Memory resources
  - Prefetch, outstanding load/ stores
  - Pinned pages or equivalent NIC needs
  - Transactional memory regions
  - Memory use (buffers)
- NIC resources
  - Collective groups
  - Routes
  - Power
- OS resources
  - Synchronization hardware
  - Scheduling
  - Virtual memory
  - Cores (dark silicon)

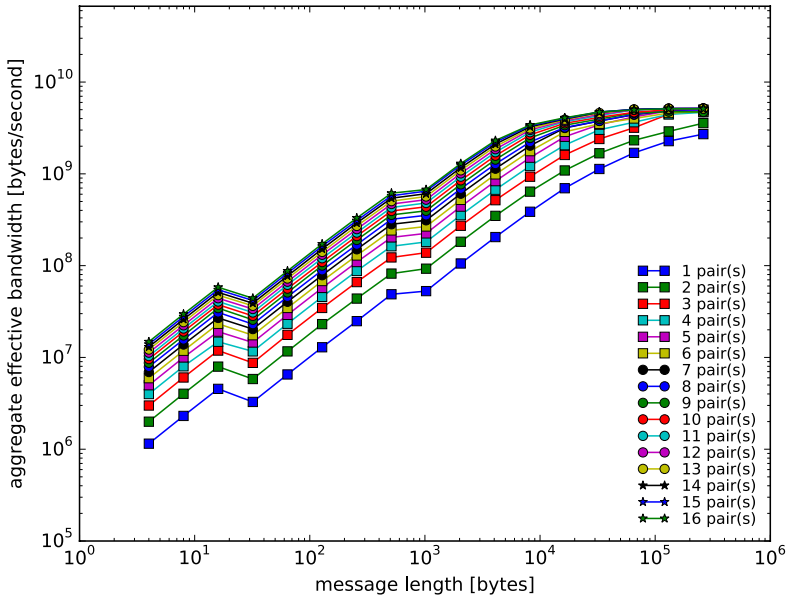
Hybrid Programming with Shared Memory

- MPI-3 allows different processes to allocate shared memory through MPI
  - MPI_Win_allocate_shared
- Uses many of the concepts of one-sided communication
- Applications can do hybrid programming using MPI or load/store accesses on the shared memory window
- Other MPI functions can be used to synchronize access to shared memory regions
- Can be simpler to program for both correctness and performance than threads because of clearer locality model

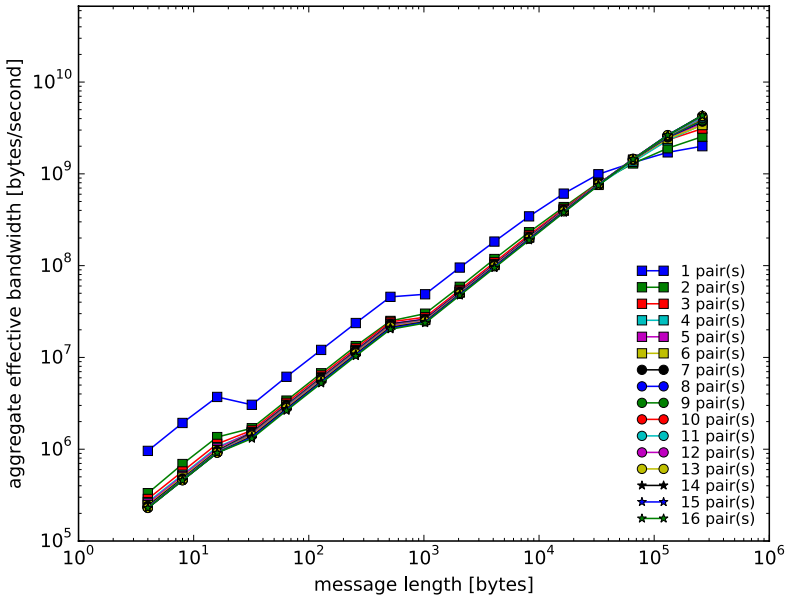
A Hybrid Thread-Multiple Ping Pong Benchmark

- In a hybrid thread-multiple approach, what if t threads communicate instead of t processes?
  - The benchmark was extended towards a multithreaded version where t threads do the ping pong exchange for a single process per node (i.e., k = 1)
  - Results for Blue Waters (Cray XE6)
  - The number t of threads and message sizes n are varied
- Results show
  - Our performance model no longer applies …
  - Performance of multithreaded version is poor
  - This is due to excessive spin and wait times spent in the MPI library
    - Not an MPI problem but a problem in the implementation of MPI

Results for Multithreaded Ping Pong Benchmark
Coarse-Grained Locking

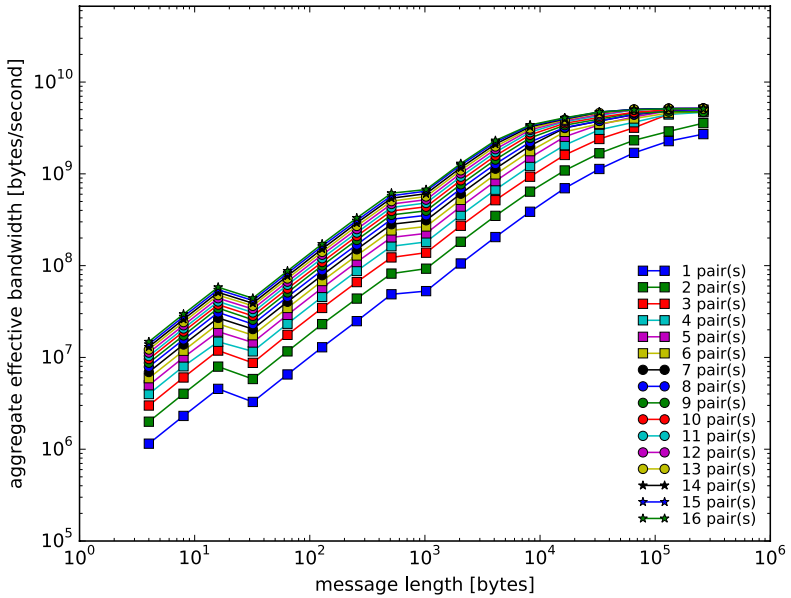


Measurements for single-threaded benchmark

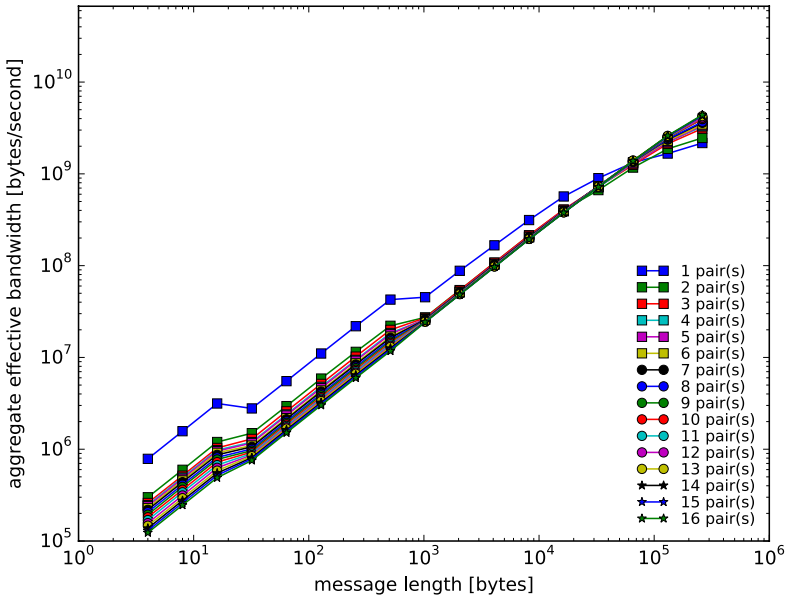


Measurements for multi-threaded benchmark

Results for Multithreaded Ping Pong Benchmark
Fine-Grained Locking



Measurements for single-threaded benchmark



Measurements for multi-threaded benchmark

Implications For Hybrid Programming

- Model and measurements on Blue Waters suggest that if a fixed amount of data needs to be transferred from one node to another, the hybrid master-only style will have a disadvantage compared to pure MPI
- The disadvantage might not be visible for very large messages where a single thread (calling MPI in the master-only style) might be able to saturate the NIC
- In addition, a thread-multiple hybrid approach seems to be currently infeasible because of a severe performance decline in the current MPI implementations
  - Again, not a fundamental problem in MPI; rather, an example of the difficulty of achieving high performance with general threads

Lessons Learned

- Achieving good performance with hybrid parallelism requires careful management of concurrency, locality
- Fine-grain approach has potential but suffers in practice; coarse-grain approach requires more programmer effort but gives better performance
- MPI+MPI and MPI+OpenMP both practical
- Concurrent processing of non-contiguous data also important (gives advantage to multiple MPI processes; competes with load balancing
- Problem decomposition and (hybrid) parallel communication performance are interdependent, a holistic approach is therefore essential

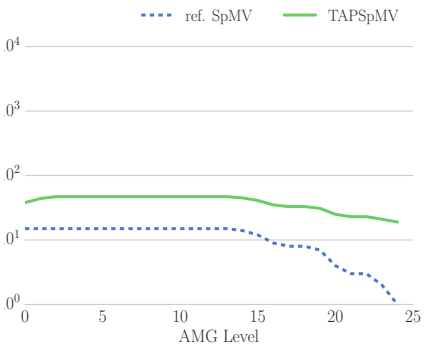
More Challenges For Extreme Scale Systems

- Simple MPI everywhere models hide important performance issues
  - Impacts algorithms – ex SpMV
- MPI implementations don't take nodes into account
  - Impacts memory overhead, data sharing
  - Process topology – Cart_create wrong API – ex nodecart
- File I/O bottlenecks
  - Metadata operations impact scaling, even for file/process (or should it be file per node?)
  - Need to monitor performance; avoid imposing too much order on operations – ex MeshIO
- Communication synchronization
  - Common "bogeyman" for extreme scale
  - But some of the best algoriitms use, e.g., Allreduce
  - Reorder operations to reduce communication cost; permit overlap
  - Ex scalable CG algorithms and implementations

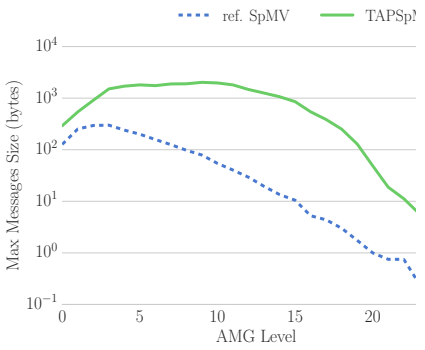
Node-Aware Sparse Matrix-Vector Product

- Sparse matrix-vector products the core to many algorithms
  - E.g., in Krylov methods and in stencil application
- "Good" mappings of processes to nodes for locality also mean that the same data may be needed for different processes on the same node
- Can significantly improve performance by trading intra-node for internode communication…
- Work of Amanda Bienz and Luke Olson

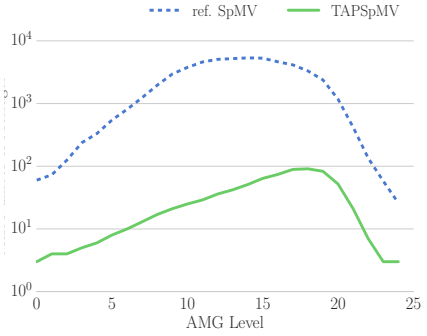
TAPSpMV Communicatio



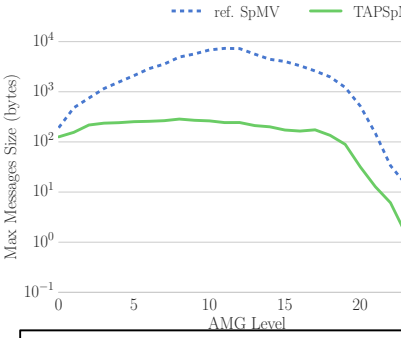
Number intra-node



Size intra-node



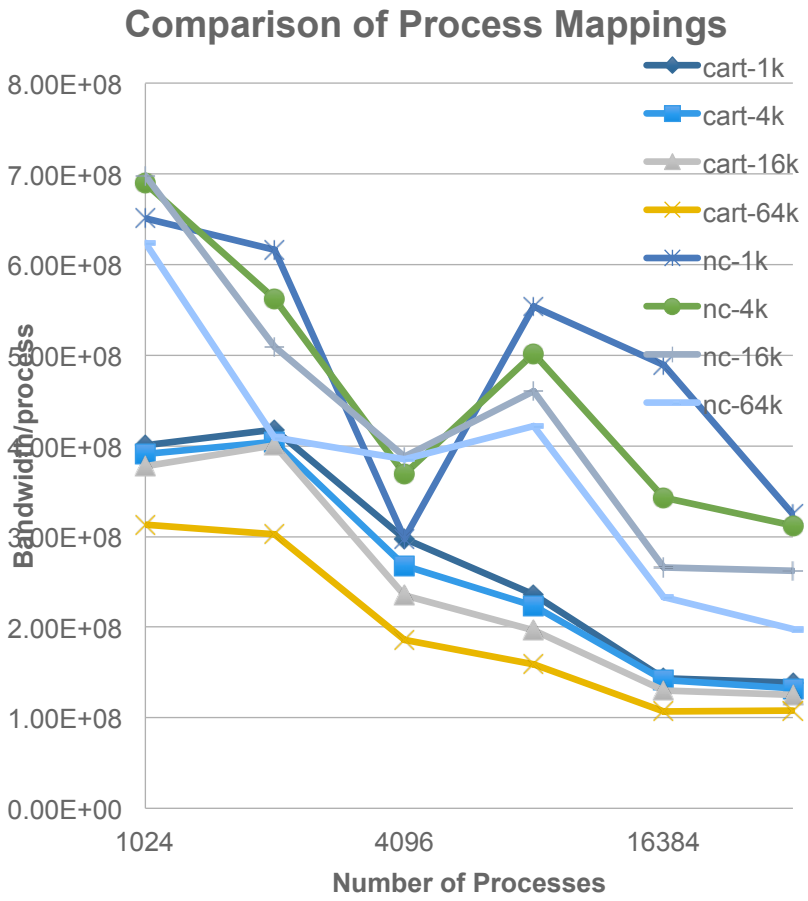
Number inter-node



Size inter-node

MPI Process Topology: The Reality

- MPI provides a rich set of routines to allow the MPI implementation to map processes to physical hardware
- But in practice, behaves poorly or ignored (allowed by the standard)
- Halo exchange illustrates
  - Cart uses MPI_Cart_create
  - Nc is a user-implemented version that taeks noes into account
  - Nc is about 2x as fast
  - Note both have scaling problems (the network topology)



IO Performance Often Terrible

- Applications just assume I/O is awful and can't be fixed
- Even simple patterns not handled well
- Example: read or write a submesh of an N-dim mesh at an arbitrary offset
- Needed to read input mesh in PlasComCM. Total I/O time less than 10% for long science runs (that is < 15 hours)
  - But long init phase makes debugging, development hard

	Original	Meshio	Speedup
PlasCom CM	4500	1	4500
MILC	750	15.6	48

- Meshio library built to match application needs
- Replaces many lines in app with a single *collective* call
- Meshio https://github.com/oshkosher/meshio
- Work of Ed Karrels

Scalable Preconditioned Conjugate Gradient Methods

- Reformulations of CG trade computation for the ability to overlap communication
- Hide communication costs and absorb noise to produce more consistent runtimes
- Must overlap allreduce with more matrix kernels as work per core decreases and communication costs increase
- Faster, more consistent runtimes in noisy environments
- Effective for simpler preconditioners and shows some speedups for more complex preconditioners without modifications
- Work of Paul Eller, “Scalable Non-blocking Preconditioned Conjugate Gradient Methods”, SC16 http://ieeexplore.ieee.org/document/7877096/

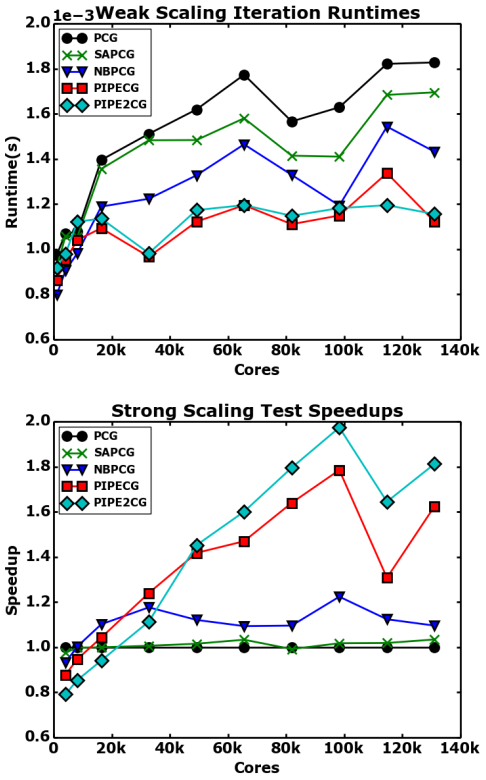


Figure: 27-point Poisson matrices with 4k rows per core (top) and 512^3 rows (bottom)

Summary

- Multi- and Many-core nodes require a new communication performance model
  - Implies a different approach to algorithms and increased emphasis on support for asynchronous progress
- Intra-node communication with shared memory *can* improve performance, but
  - Locality remains critical
  - Fast memory synchronization, signaling essential
    - Most (all?) current MPI implementations have very slow intra-node MPI_Barrier.
- Many algorithms, data structures, and implementations need to be re-examined for multicore nodes

Thanks!

- Philipp Samfass
- Luke Olson
- Pavan Balaji, Rajeev Thakur, Torsten Hoefler
- ExxonMobile Upstream Research
- Blue Waters Sustained Petascale Project, supported by the National Science Foundation (award number OCI 07–25070) and the state of Illinois.
- Cisco Systems for access to the Arcetri UCS Balanced Technical Computing Cluster