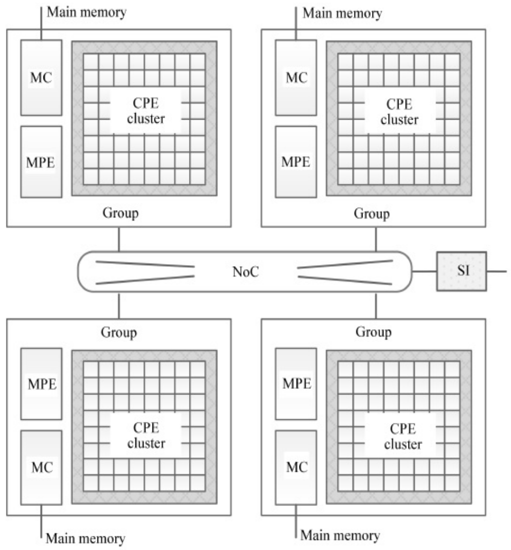
MPI+X on The Way to Exascale

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

NCSA

Some Likely Exascale Architectures



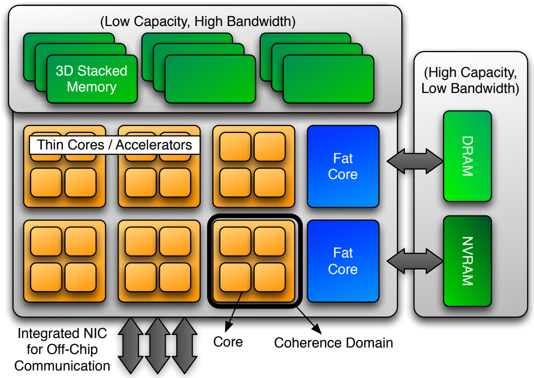
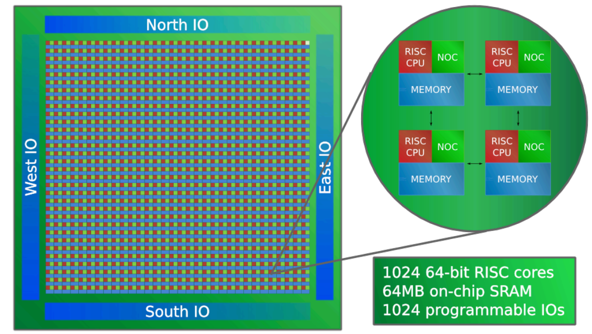


Figure 2.1: Abstract Machine Model of an exascale Node Architecture



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

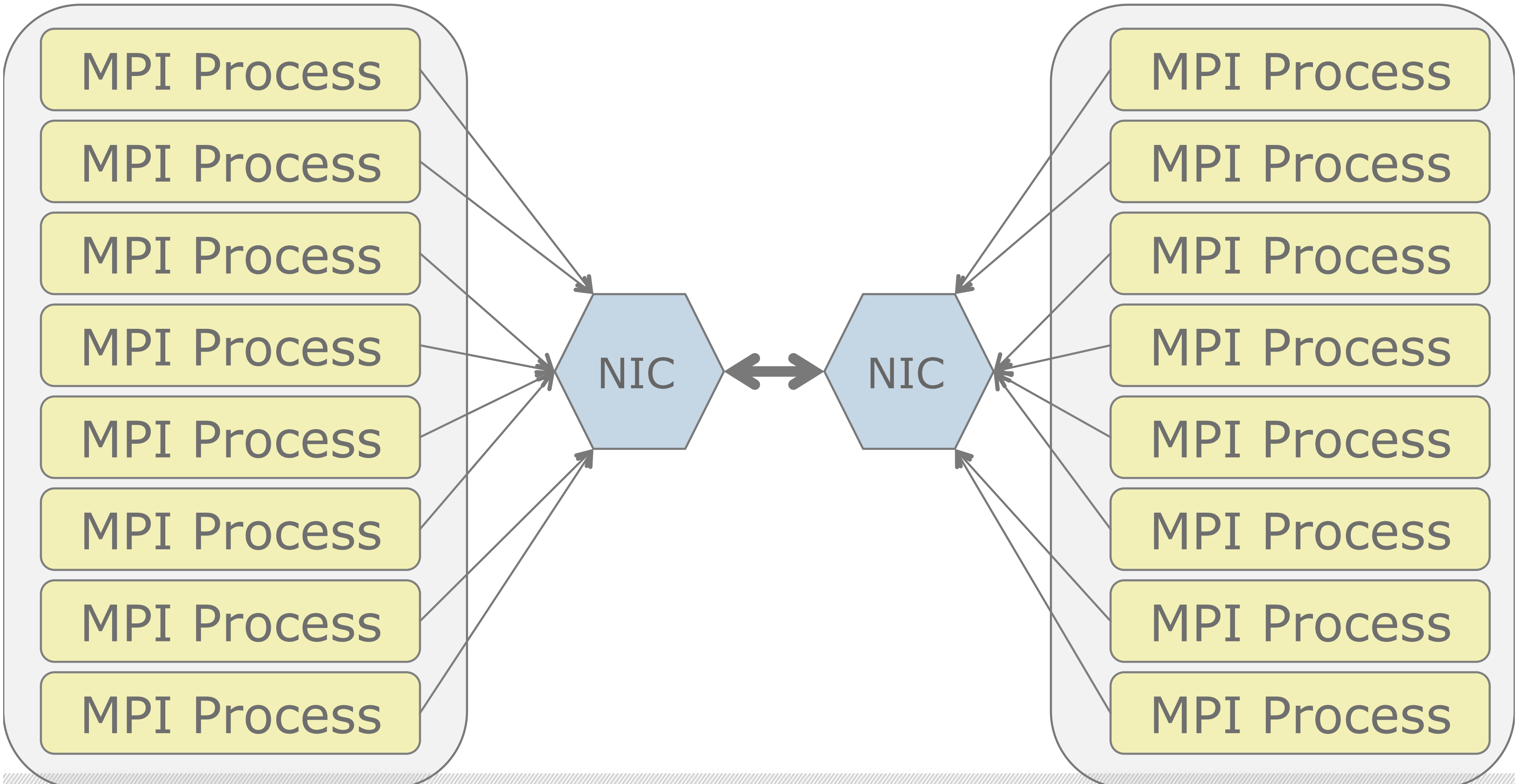
Applications Still Mostly MPI-Everywhere

- "the larger jobs (> 4096 nodes) mostly use message passing with no threading." – Blue Waters 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…

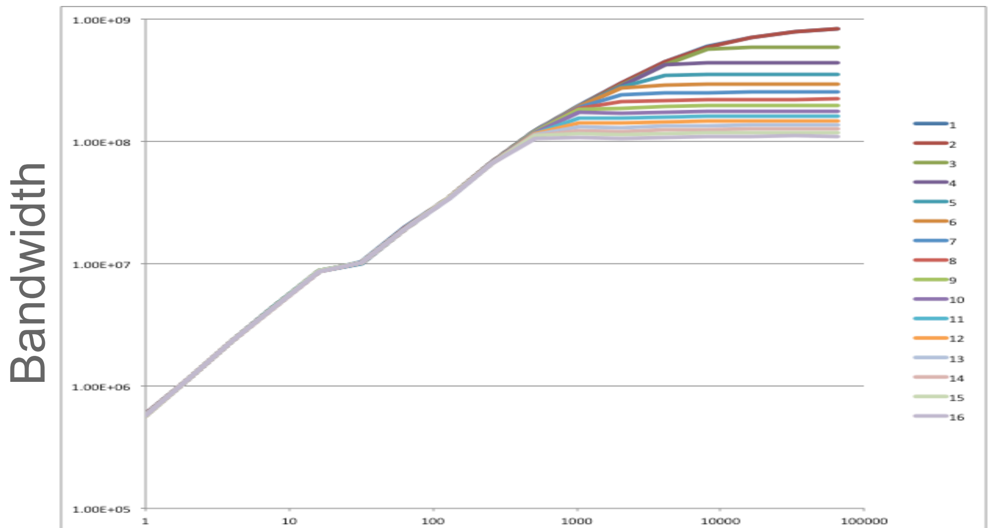
SMP Nodes: One Model

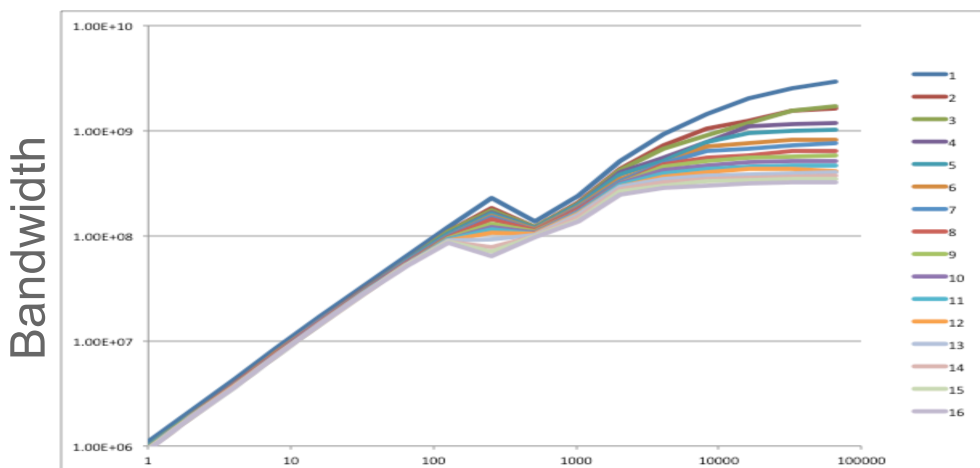


Classic Performance Model

- 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

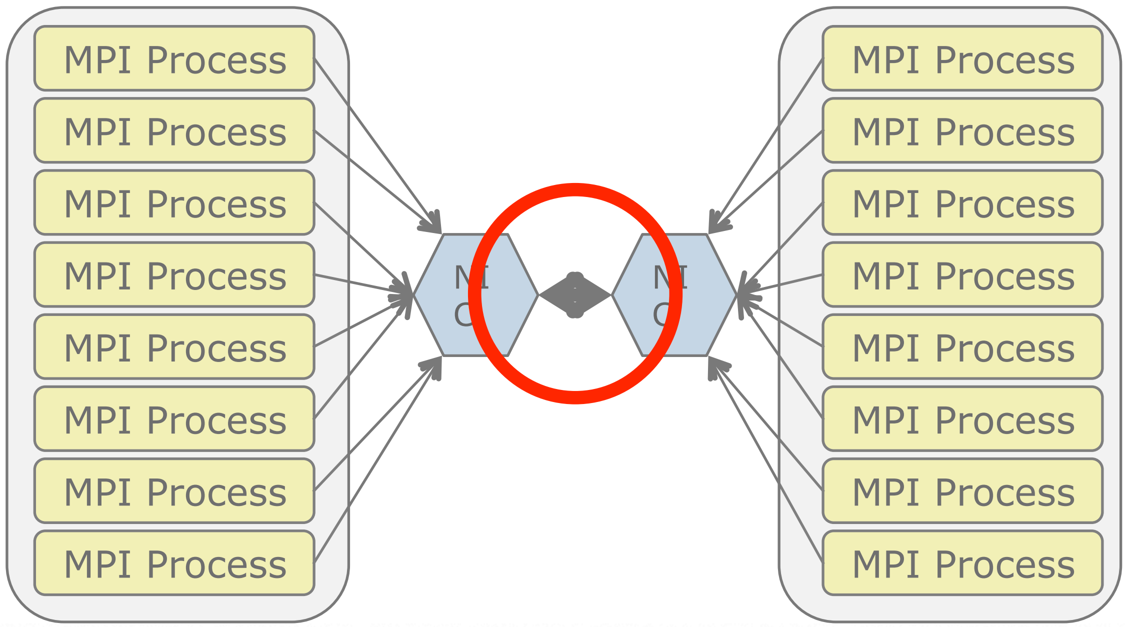




- Ping-pong between 2 nodes using 1-16 cores on each node
- Top is BG/Q, bottom Cray XE6
- "Classic" model predicts a single curve – rates independent of the number of communicating processes

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

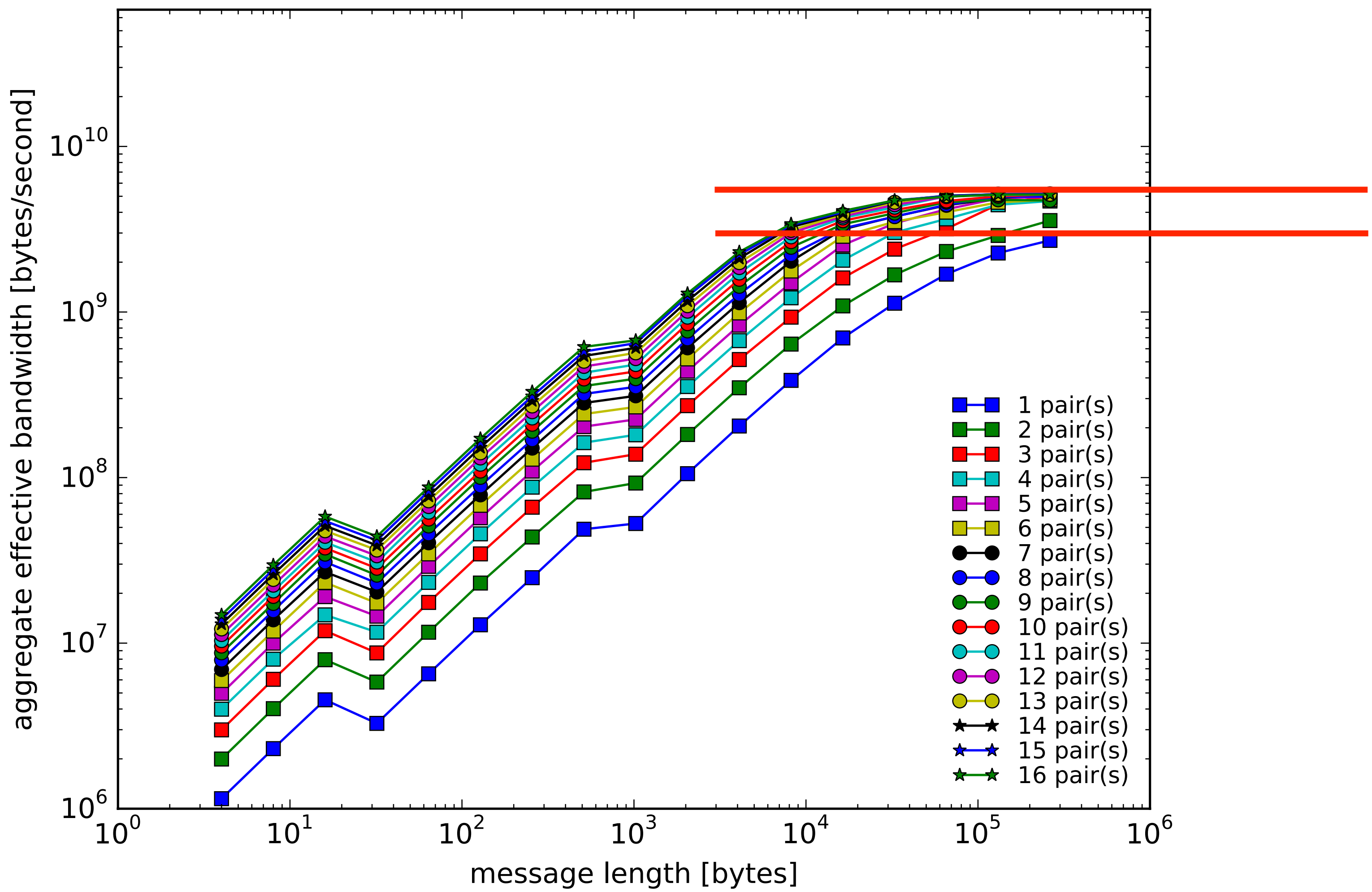
A Slightly Better Model

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

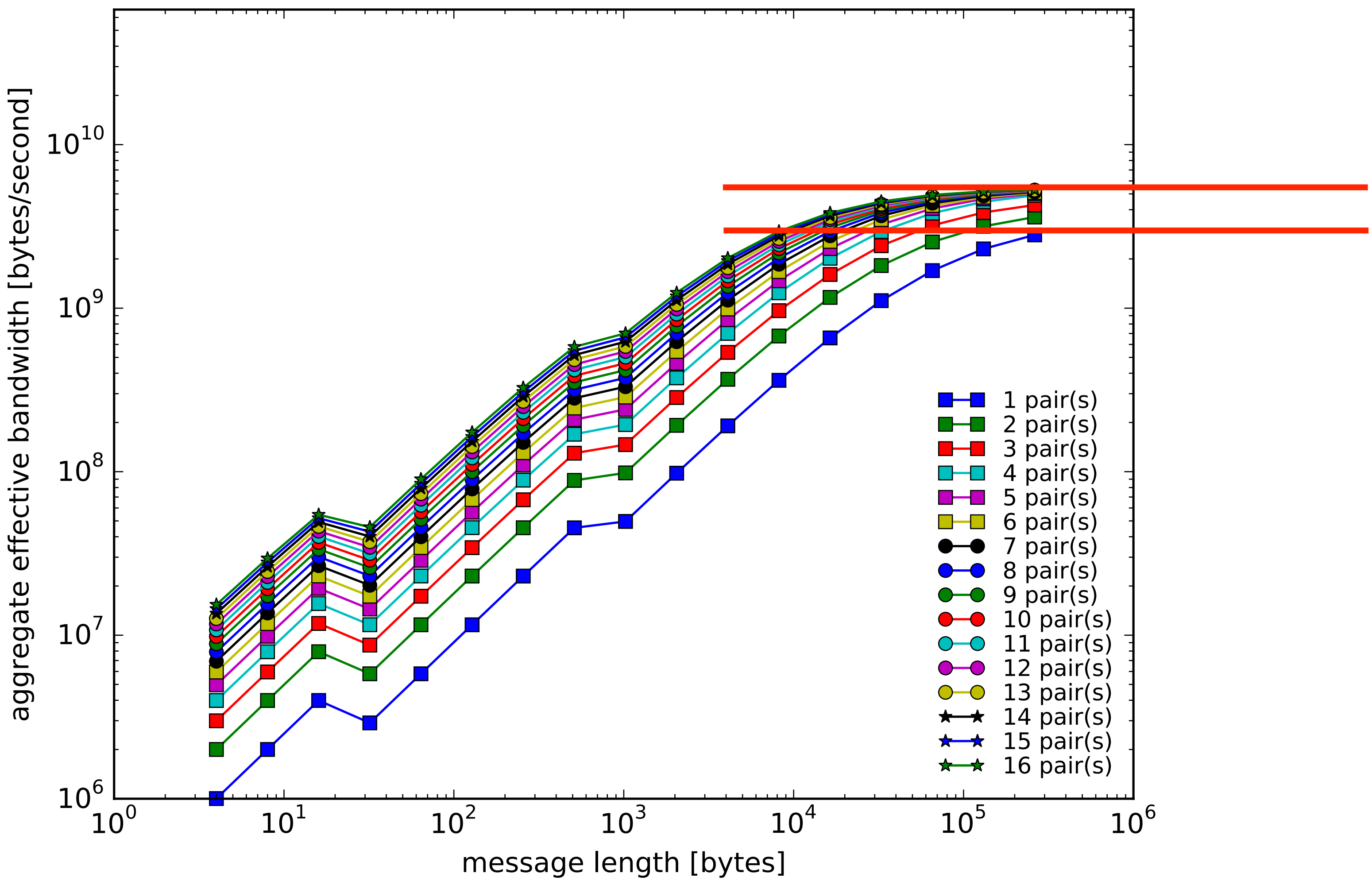
How Well Does this Model Work?

- 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

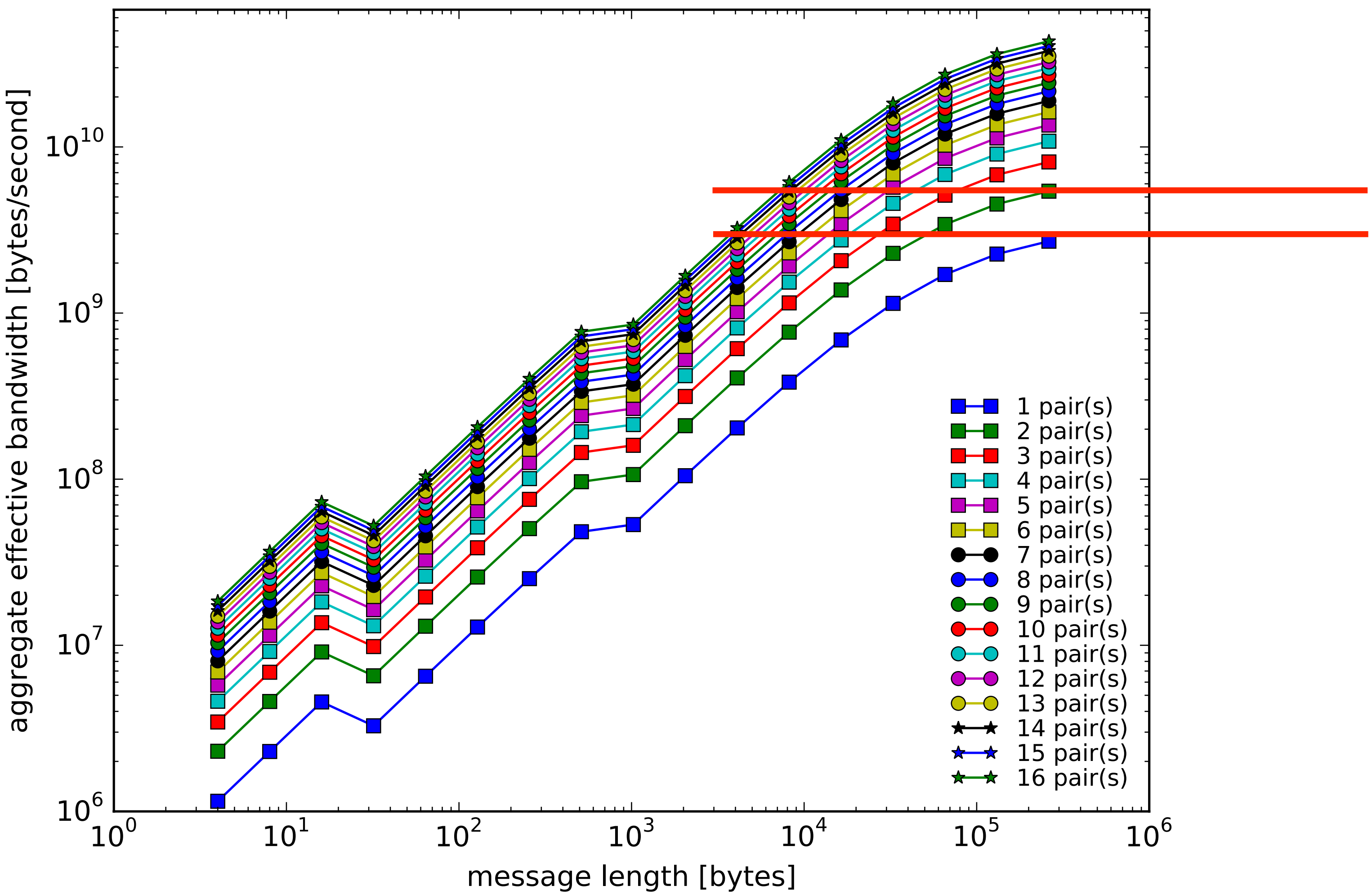
Cray: Measured Data



Cray: 3 parameter (new) model



Cray: 2 parameter model



Implications

- Simple "BSP" style programming will often be communication limited
- MPI supports many more flexible and general communication approaches
  - But users must use them
- (Relatively) Simple
  - Use communication/computation overlap
    - MPI must implement at least limited asynchronous progress
  - Exercise care in mapping MPI processes to cores/chips/nodes
- Use one-sided programming
  - Mostly non-blocking by design
  - MPI Forum continuing to look at extensions, such as one-sided notification and non-blocking synchronization
- Use lightweight threads with over-decomposition
  - Let thread scheduler switch between communication and compute

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)

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
    - Implementation is tricky, for example:
    - Most (all?) current MPI implementations have very slow intra-node MPI_Barrier.

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