Programming for Optimal MPI Performance on LC's Linux / Quadrics Clusters

Adam Moody
moody20@llnl.gov
Development Environment Group

May 23, 2005

UCRL-PRES-212679

This work was performed under the auspices of the U.S. Department of Energy by University of California, Lawrence Livermore National Laboratory under Contract W-7405-Eng-48.

Outline

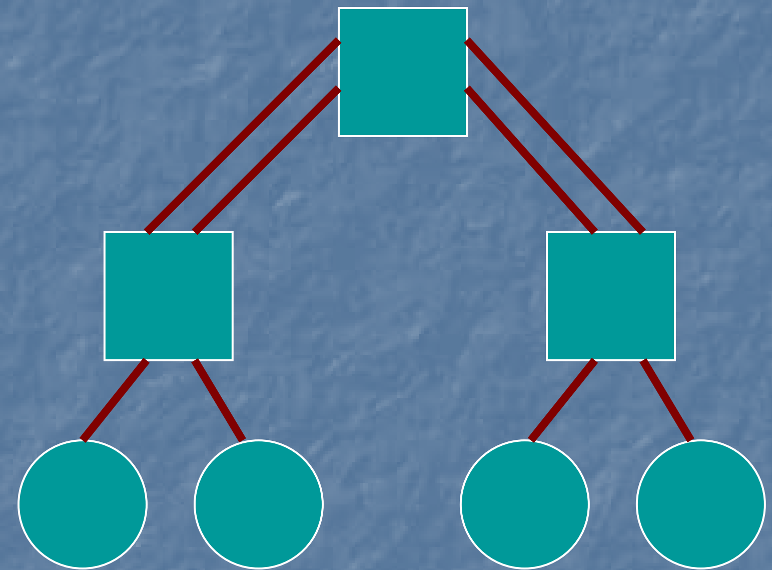
- Overview
  - System Network Architecture
  - The Quadrics Interconnect
- Programming for Optimal MPI
- Tuning the Runtime Environment
- Wrap-up

System Network Architecture

- Thunder
  - 4 Itanium2's per node
  - PCI-X: 64-bit @ 133 MHz ~ 1066 MB/sec
  - $QsNet^{II}$: 2.5 usec, 900 MB/sec
- MCR
  - 2 Xeon's per node
  - PCI: 64-bit @ 66 MHz ~ 533 MB/sec
  - QsNet: 4.7 usec, 320 MB/sec

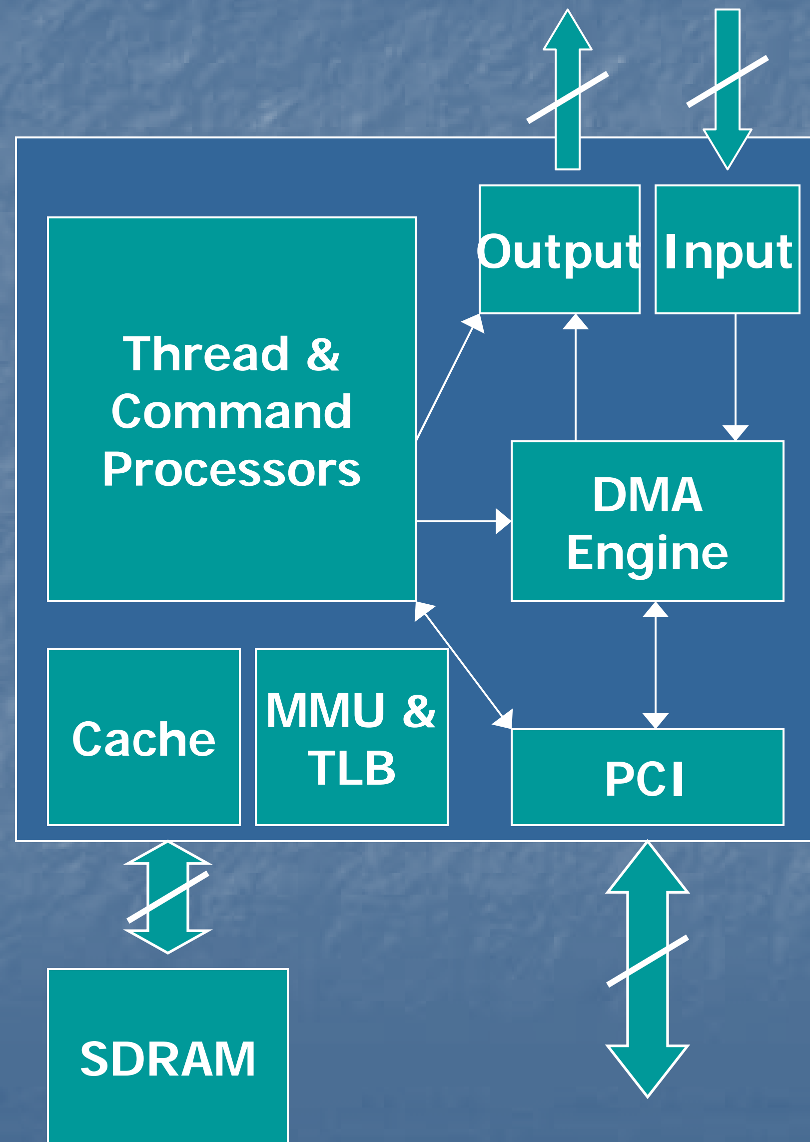
The Quadrics Interconnect

- Elite (the switch)
  - Fat-tree architecture
  - Full duplex links
  - Source-based, wormhole routing
  - Hardware support for packet broadcasts, remote tests, and reliable transmission



The Quadrics Interconnect

- Elan (the NIC)
  - Remote DMA support
    - OS-bypass, zero-copy
    - Local MMU & TLB
  - Communication Offload
    - Programmable RISC threaded processor
    - 64 MB local SDRAM



The Quadrics Interconnect

- Low latency
- High bandwidth
- Full duplex
- Communication offload
- Hardware support for broadcast
- Hardware support for reliable transmission

Outline

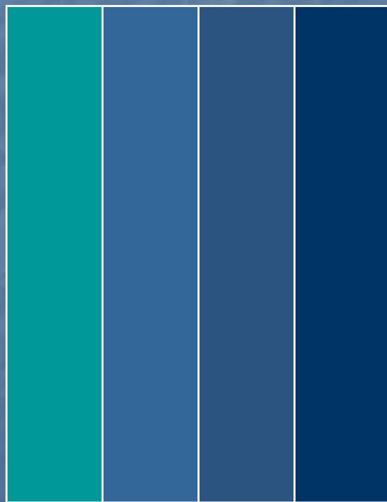
- Overview
- Programming for Optimal MPI
  - Design to minimize communication
  - Many small messages kill performance
  - Take advantage of offloading, zero-copy, OS-bypass
- Tuning the Runtime Environment
- Wrap-up

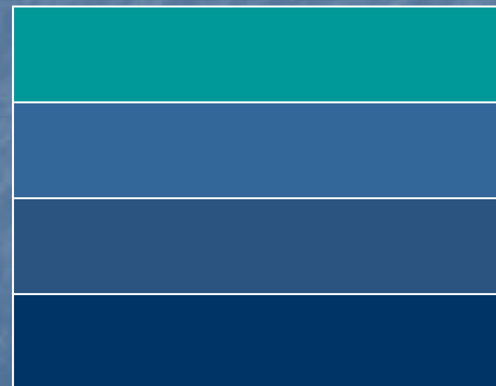
Programming for Optimal MPI

- Design to minimize communication
  - Map data to a process so as to minimize its number of communication partners first, message size second
  - Avoid global communication operations, divide processes into sub-communicators where possible
  - Consider hybrid MPI / OpenMP
  - Always use collectives, but know that gather, scatter, and alltoall are inherently non-scalable
- Many small messages kill performance
- Take full advantage of offloading, zero-copy, OS-bypass

Data Mapping

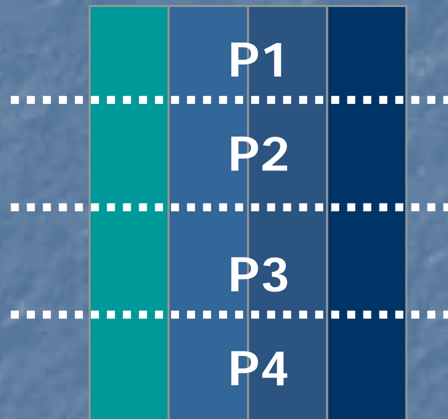
- For example, say our problem requires a matrix transpose. What is the best way (wrt MPI) to assign the matrix to four processors?

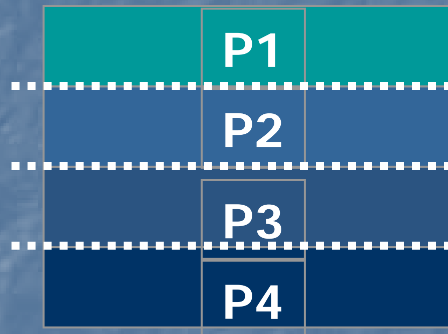




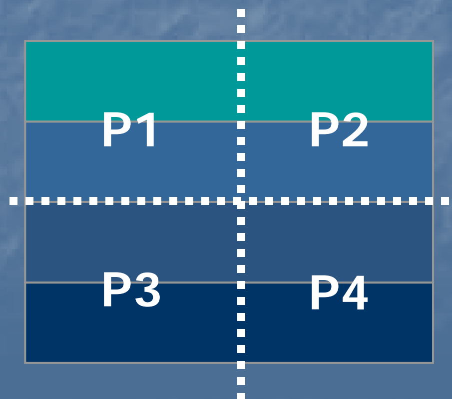
Data Mapping

- In general, it's better to send a few big messages rather than a lot of little ones.
  - Reduce number of messages first, message size second





- Requires MPI_Alltoall()
  - P^2 messages
  - Less scalable



- Requires MPI_Send/Recv()
  - P messages
  - More scalable

Global communication and Sub-communicators

- If possible, avoid design choices that require processes to synchronize or exchange data with all other processes (e.g., collectives on MPI_COMM_WORLD)
- Instead, favor algorithms that divide processes into subsets via sub-communicators
  - It is ok to overlap communicators and assign a process to more than one
- Create sub-communicators early on and reuse them
  - Several global collectives are called to create a communicator, so creating one to do just a few operations before deallocating it may be self-defeating

Hybrid MPI / OpenMP Advantages (wrt MPI-only)

- Sharing data between processors on same node
  - MPI – data copied from one address space to another
    - Latency: microsecs; Bandwidth: memory-copy speed
  - OpenMP – shared address space, threads synchronize
    - Latency: nanosecs; Bandwidth: infinite (data doesn't move)
- Sharing data between processors on diff. nodes
  - MPI-only: multiple MPI procs / node, one network link
    - MPI processes receive fraction of network bandwidth
  - Hybrid: one MPI proc / node, one network link
    - MPI processes receive full network bandwidth
    - Fewer processes, bigger message sizes – generally good for scalability

Hybrid MPI / OpenMP Disadvantages

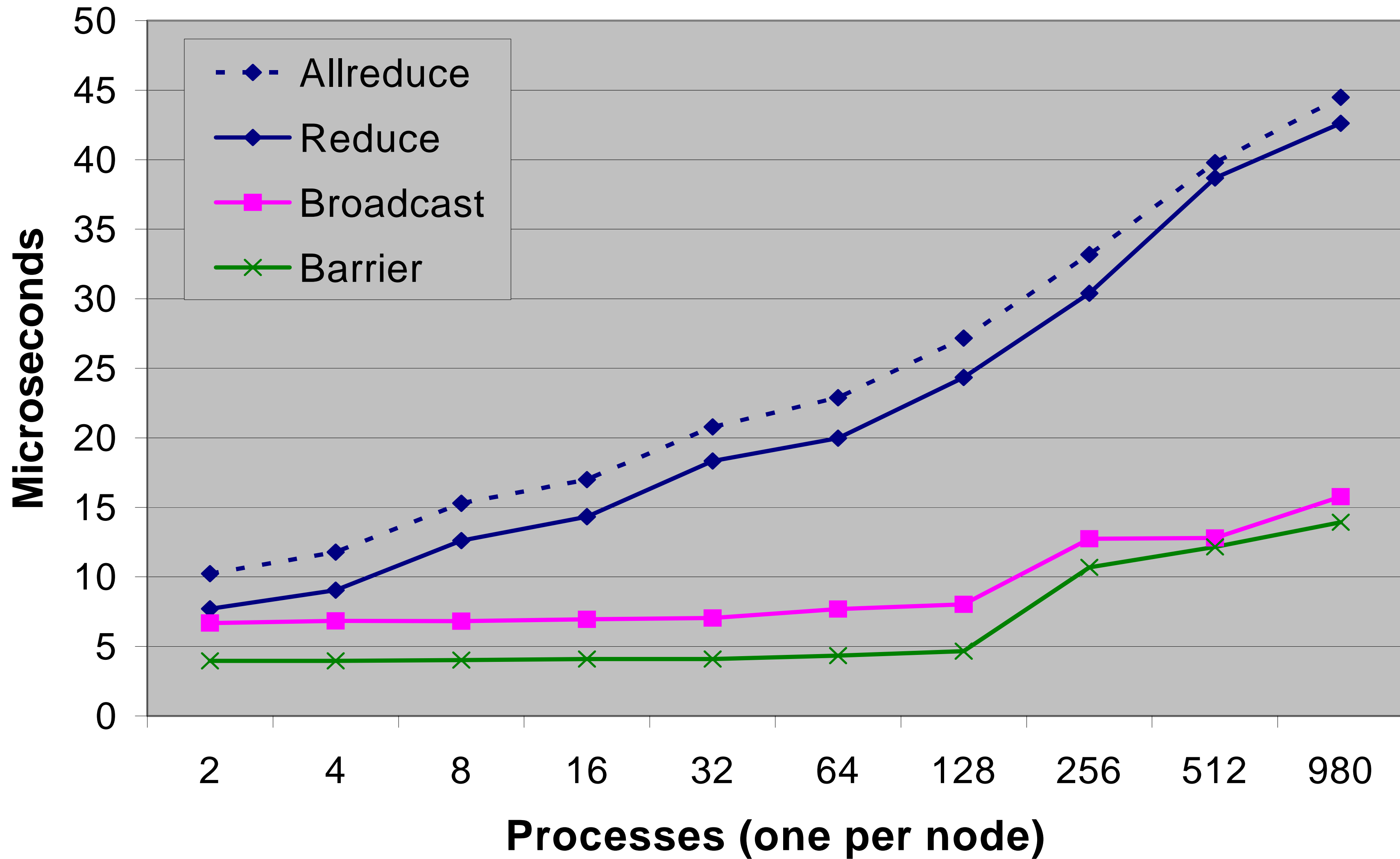
- OpenMP may be difficult to implement depending on your algorithm
- OpenMP maintenance
  - need thread-safe libraries, including 3rd party
  - must be built with thread-safe compilers
- OpenMP performance
  - depends on support and quality of compiler
  - thread-processor affinity
  - overhead may outweigh benefits

MPI Collectives

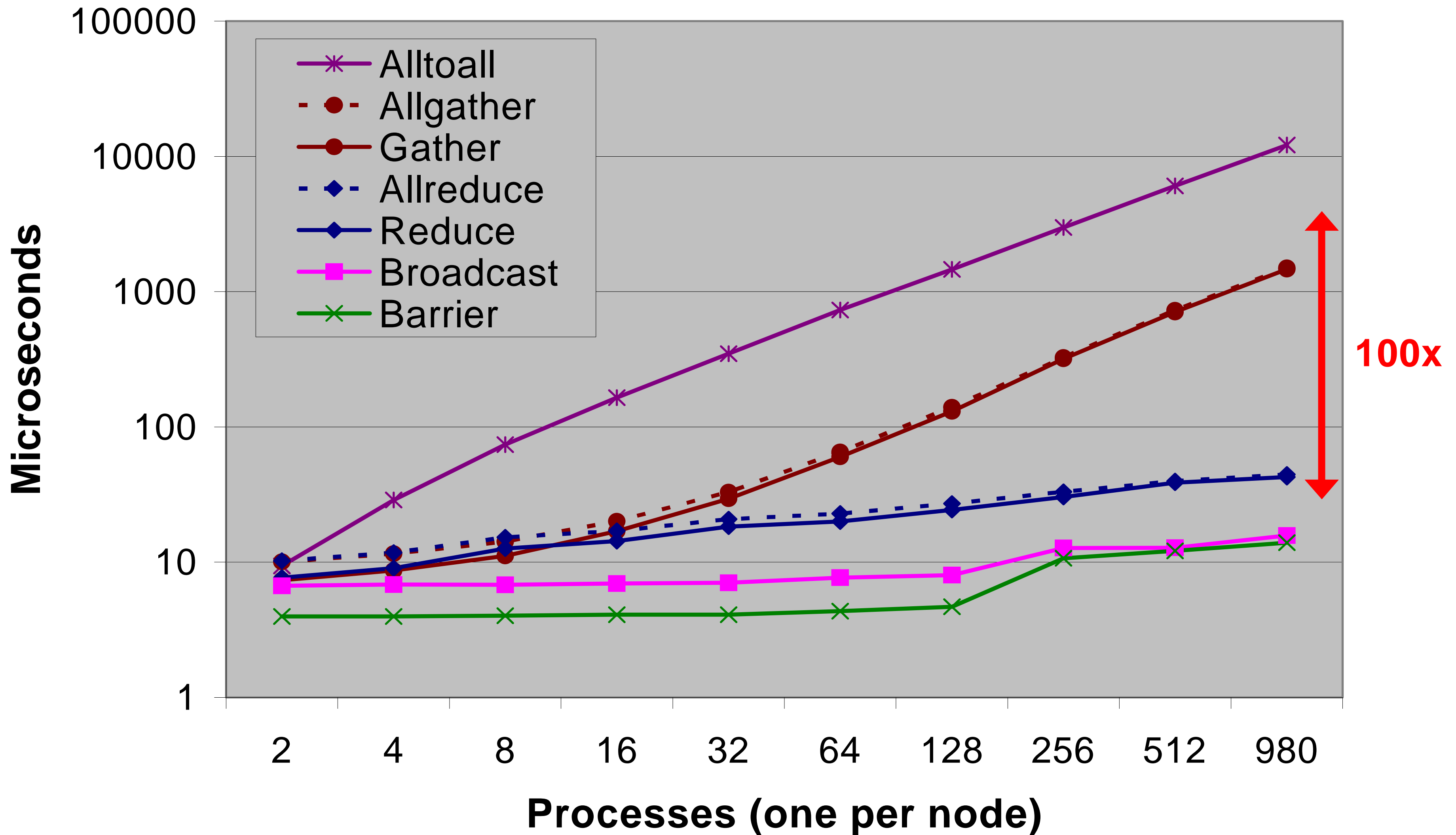
- MPI vendors tune collectives optimally for underlying hardware
  - Leverage their work; use interconnect to its fullest
- Common collectives to be aware of:
  - Barrier
  - Broadcast
  - Reduce, Allreduce
  - Gather, Allgather, Gatherv, Allgatherv
  - Scatter, Scatterv
  - Alltoall, Alltoallv
- If you're unfamiliar with these, take a few minutes and read about them in the MPI standard – you may find them useful

Collective Scalability on Quadrics

- For P processes and message size M:
  - Barrier, Broadcast — O(P,M) ~ M
  - Reduce, Allreduce — O(P,M) ~ log(P)*M
  - Gather, Allgather, Scatter — O(P,M) ~ P*M
  - Alltoall — O(P,M) ~ P*M + contention
- Quadrics hardware provides superb scalability of Barrier, Broadcast, Reduce, Allreduce
- Gather, Allgather, Scatter, and Alltoall are inherently non-scalable
  - these will bite you in large-scale global operations



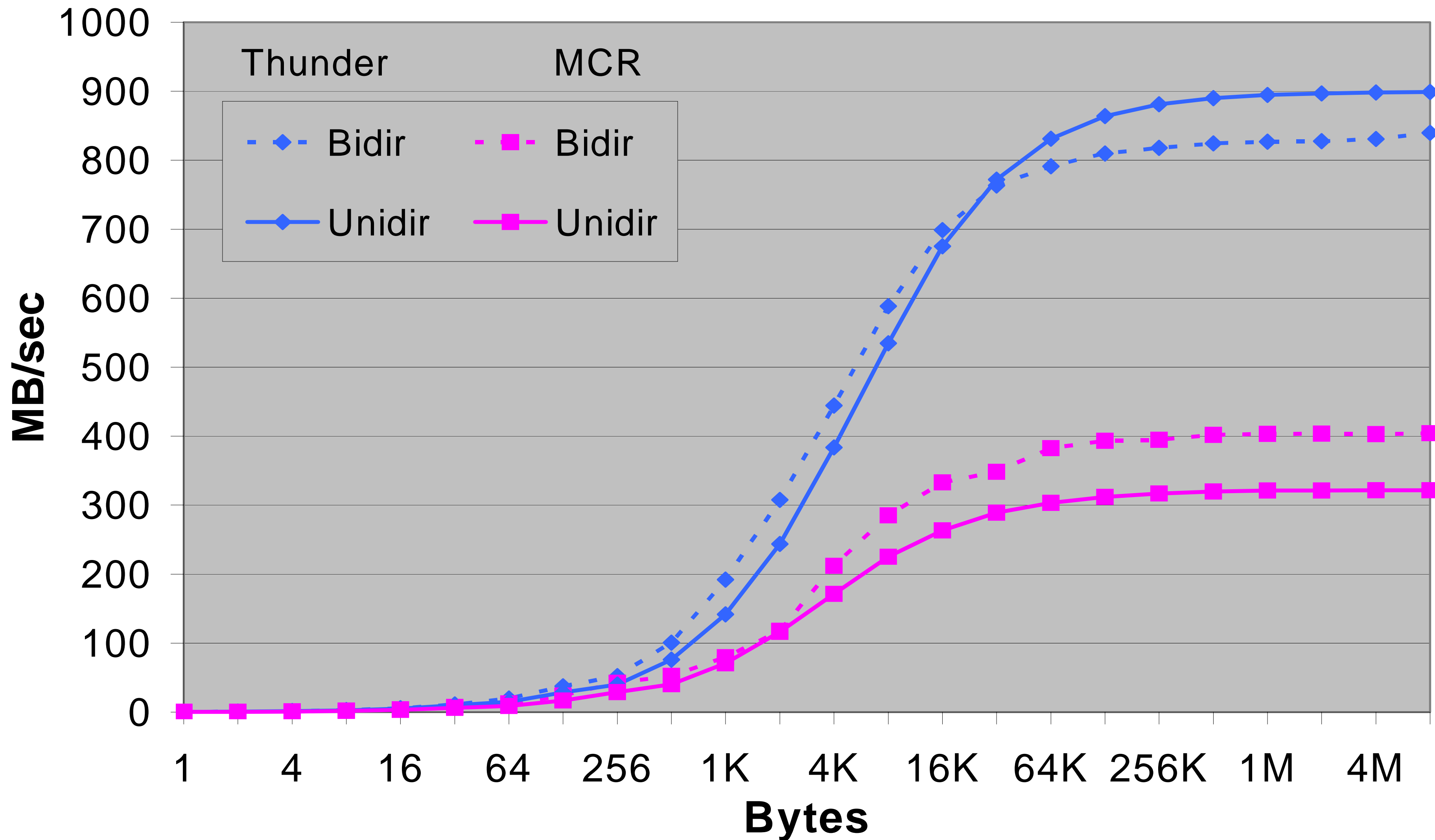
On Thunder,
Barrier and Broadcast in nearly constant time
Reduce / Allreduce on 980 processes takes only 5-6 times longer than 2 procs.



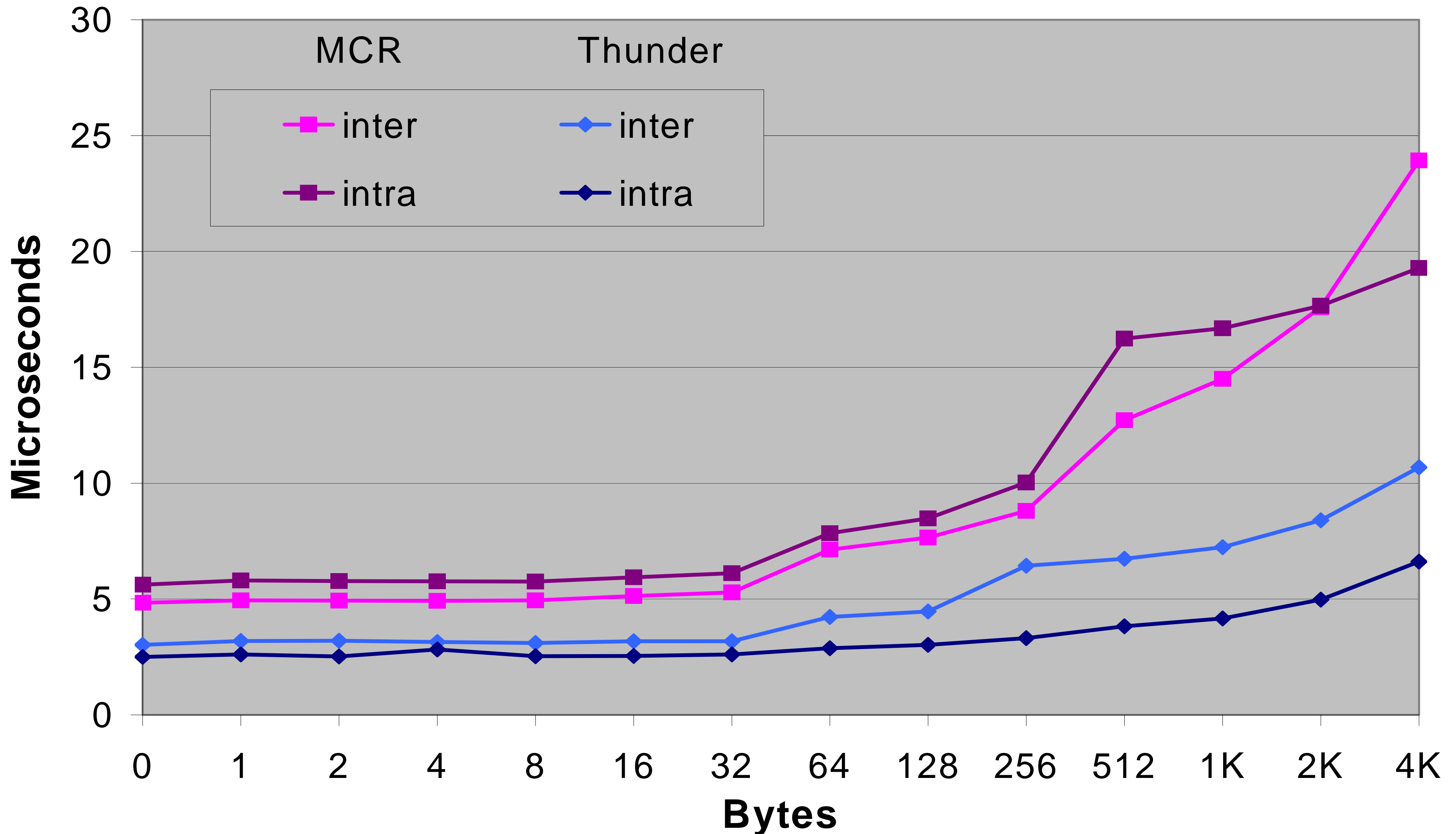
But,
Gathers and Alltoall are 100 times slower
(inherent limits in communication pattern, not interconnect or implementation)

Programming for Optimal MPI

- Design to minimize communication
- Many small messages kill performance
  - Pack them into large messages
  - Eliminate them via collectives
- Take advantage of offloading, zero-copy, OS-bypass



E.g., say we have four chunks of 1024-byte data to send to the same destination
Need to push a total of 4K of data through network
Packing into one 4K message delivers twice the bandwidth of four 1K messages
Same amount of data @ 2x the bandwidth → 1/2 the time



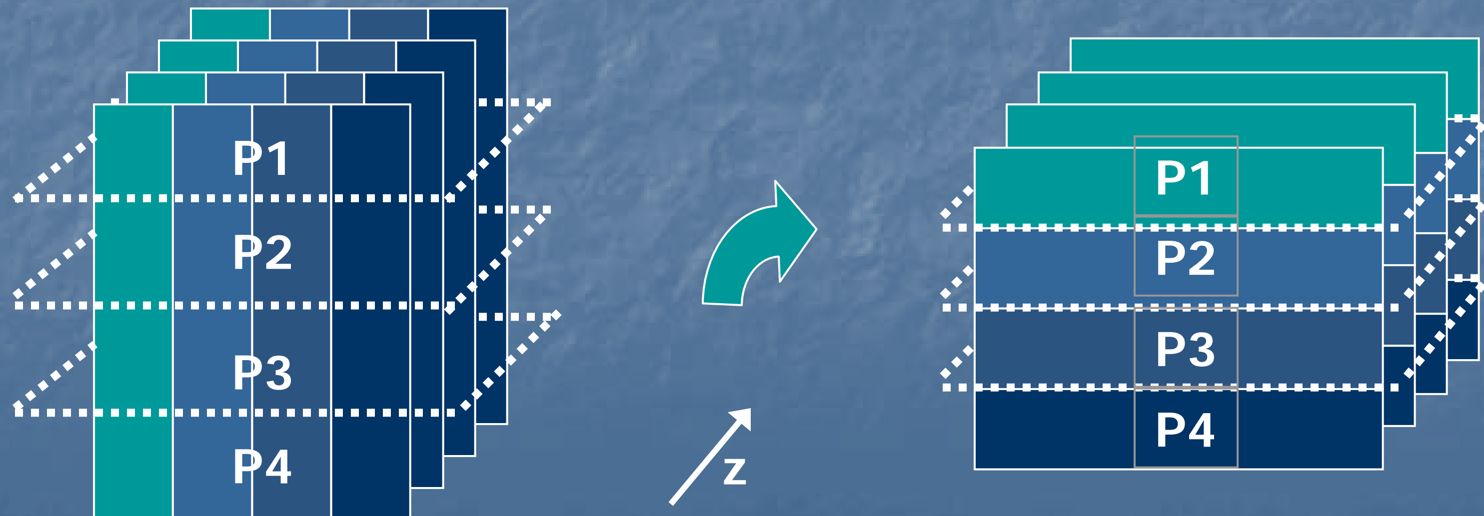
The effect of packing is more extreme for very small messages.
On Thunder, 4096 bytes can be sent between nodes in only 4x the time it takes to send just 1.

Packing Small Messages

- Packing comes with a cost
  - Memory copies required to pack and unpack the data
- Packing pays off so long as the memory copies cost less than the savings from higher bandwidth
  - Switch point occurs just before the bandwidth curve levels out
  - If your message sizes are well below this point, investigate packing

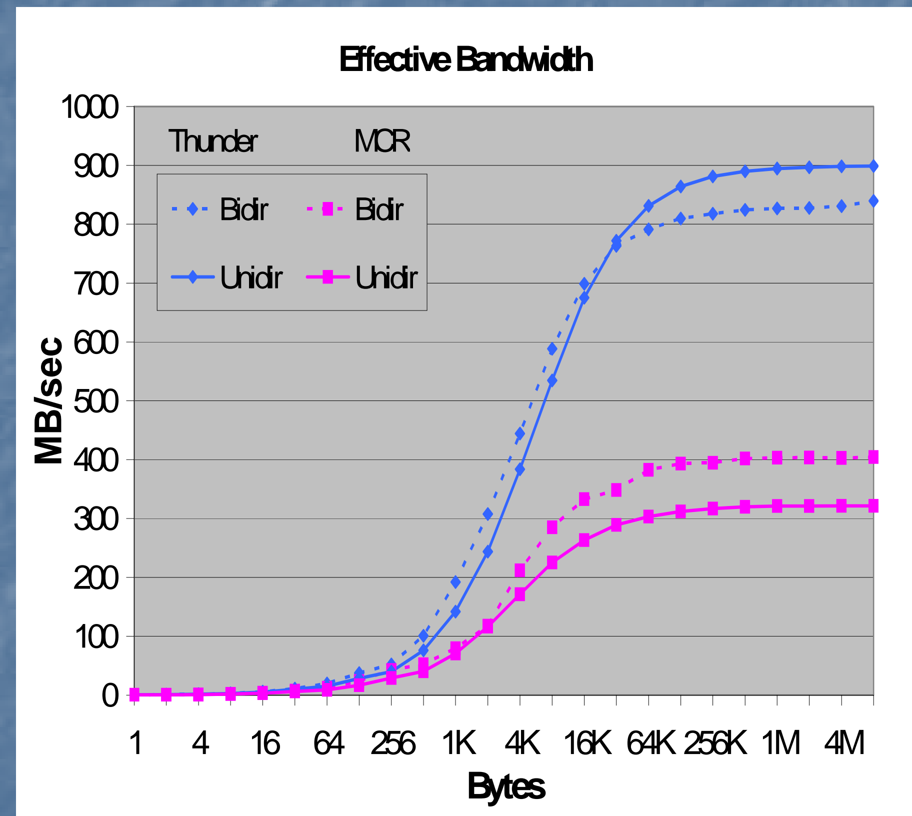
Example: Packing Small Messages

- User needed 2D transpose on 3D data set
  - Requires MPI_Alltoall()
- Original Implementation
  - Transpose each plane in z-dim, one at at time
  - Fixed problem size scaled well to tens of processors, suffered severe bottle neck at hundreds



Example: Packing Small Messages

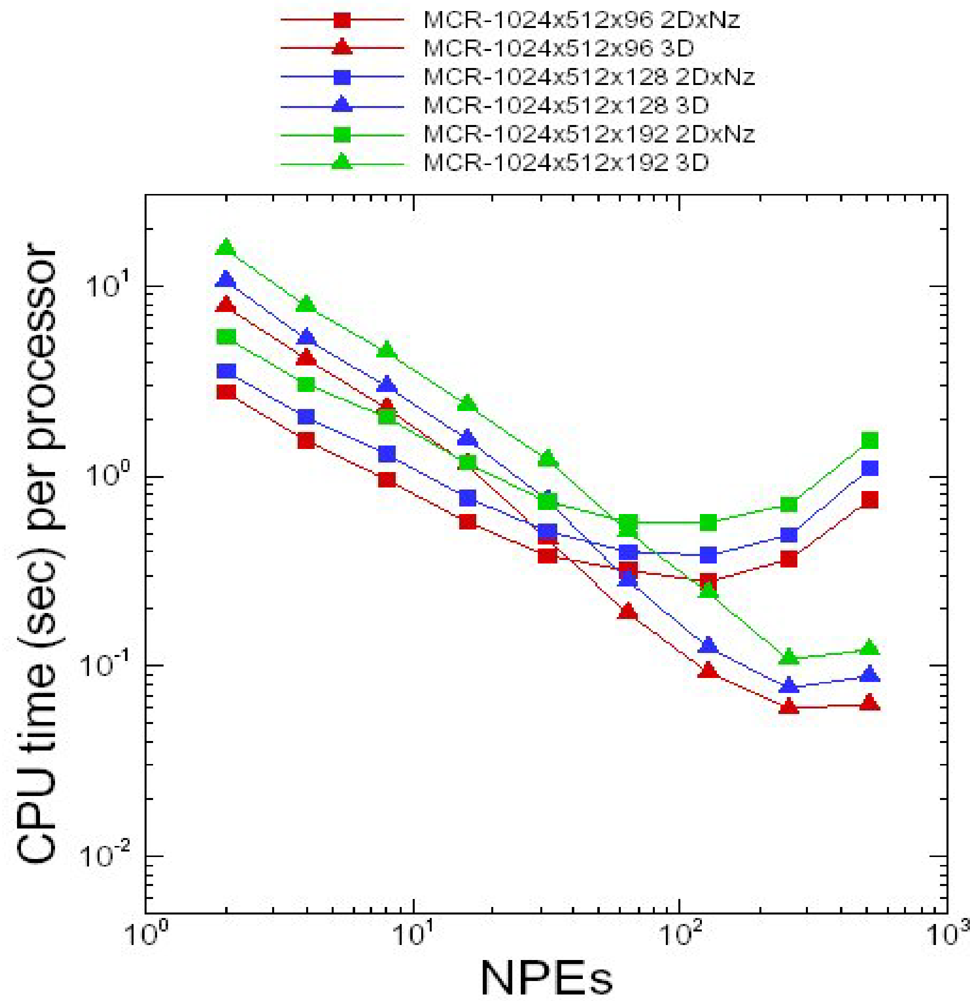
- Why did the scaling fall off?
  - When scaling a fixed-size problem to P processes, the message size in MPI_Alltoall() scales as $1/P^2$
  - Performance quickly falls off as the message size begins sliding down the bandwidth curve



Example: Packing Small Messages

- Modified Implementation
  - Pack all z-dim data in one message, perform single MPI_Alltoall(), unpack z-dim data
  - Used MPI Derived Datatypes so MPI did all packing / unpacking for us
  - Since there were 100-200 planes in the z-dim, this increased message size by 100-200 times
  - Performance returned as we shot back up the bandwidth curve

Per Processor Time for Matrix Transpose vs. Number of Processors



Packed Transpose (aka 3D)

- Ran slower at small scale, where packing / unpacking introduce overhead but no savings
- Ran 10-20x faster at large scale
- Achieved 2x speedup in application

Example 2: Packing Small Messages

- User needed to write distributed arrays in output step
  - Requires gather to root for formatting
- Original Implementation
  - For each variable, for each process, root sends "go ahead" message then receives data via MPI_Send/Recv()
  - Fixed problem size scaled well to hundreds of processes, suffered severe bottleneck at thousands

Example 2: Packing Small Messages

- Why did the scaling fall off?
  - When scaling a fixed-size problem to P processes, the message size scales as 1/P
  - Performance quickly falls off as the message size begins sliding down the bandwidth curve
- Modified Implementation
  - Pack pieces of distributed data arrays into a single message, gather to root, then unpack
  - Since there were ~70 such variables, this increased message size dramatically
  - Performance for gather increased by 10-20x
  - Achieved 2x speedup in application

Eliminating Small Messages

- Collectives eliminate small messages by packing across process boundaries
  - Data from processes on the same node can be combined into a node-wide message in shared memory
  - Tree-based algorithms combine messages from multiple children into a single message for parent
  - Hardware-based broadcast simultaneously delivers a single message to multiple processes
- Look for places to substitute collectives in place of "for" loops
  - Especially relevant to legacy codes originally designed for a small number of processes

Example: Eliminating Small Messages

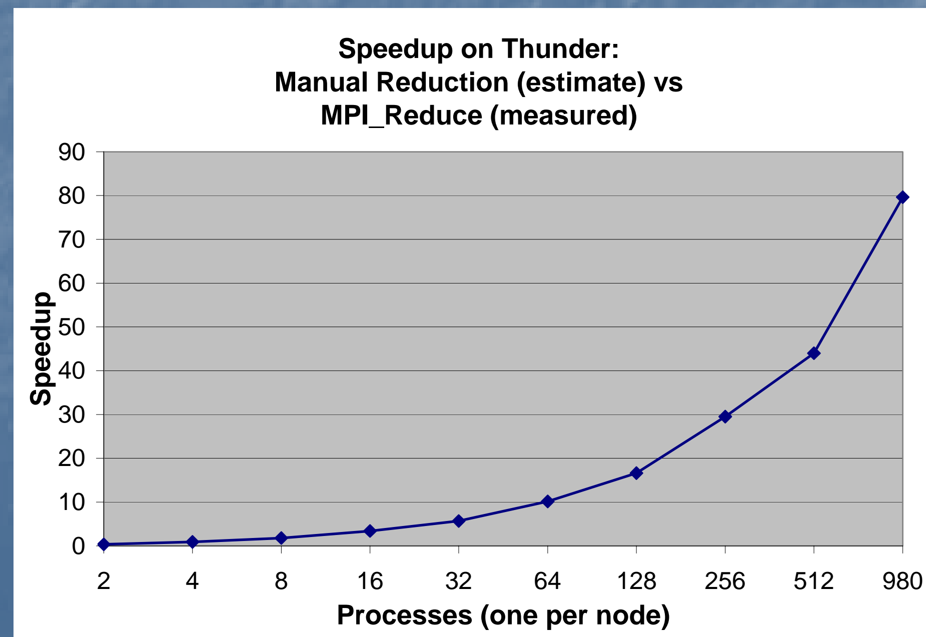
- User needed to compute global sum of some distributed data
- Original Implementation
  - For each process, root receives data via MPI_Send/Recv() and adds to running sum
  - Became severe bottleneck at thousands of processes

```
if (myid == 0)
   sum = my_data
   for i = 1 to (P-1)
      MPI_Recv(data from i)
      sum = data + sum
else
   MPI_Send(my_data to 0)
```

Example: Eliminating Small Messages

- Modified Implementation
  - Replaced manual reduction with call to MPI_Reduce()
  - Performance for reduction increased by 100-200x
  - Achieved 3x speedup in application

```
MPI_Reduce(my_data, sum, MPI_SUM, root=0)
```



Programming for Optimal MPI

- Design to minimize communication
- Many small messages kill performance
- Take advantage of offloading, zero-copy, OS-bypass
  - Use Isend/Irecv(), use collectives – NIC will assume tasks, freeing the main processor
  - Pre-post receives to avoid unexpected message buffering
  - Touch send and receive buffers to avoid page faults

Programming for Optimal MPI Summary

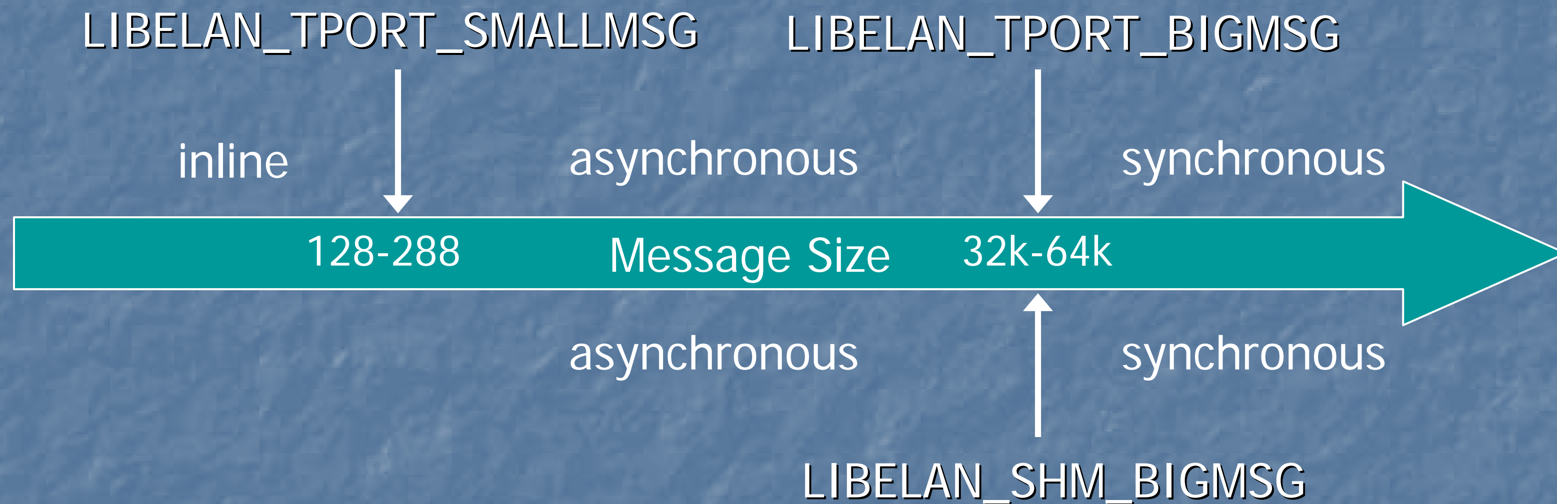
- Design to minimize communication
  - Map data to processors so as to minimize communication
  - Avoid global operations, use sub-communicators
  - Consider hybrid MPI / OpenMP
  - Always use collectives, but know that gather/scatter/alltoall are inherently non-scalable
- Many small messages kill performance
  - Pack them into large messages
  - Eliminate them via collectives
- Take full advantage of offloading, zero-copy, OS-bypass
  - Use Isend/Irecv(), use collectives – NIC will assume tasks, freeing the main processor
  - Pre-post receives to avoid unexpected message buffering
  - Touch send and receive buffers to avoid page faults

Outline

- Overview
- Programming for Optimal MPI
- Tuning the Runtime Environment
  - Tweaking algorithm switch points
  - Block vs. cyclic process distribution
  - Contiguous nodes
- Wrap-up

Tweaking Algorithm Switch Points

- Point-to-point Message Protocols



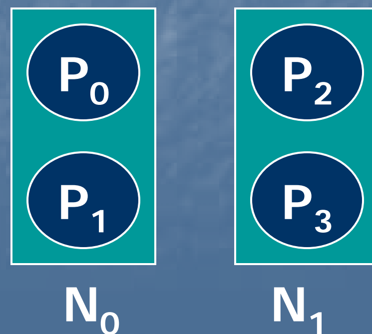
Tweaking Algorithm Switch Points

- "Best" values for switch points vary with application and problem size
  - If you send lots of messages sized near these default boundaries, experiment with tweaking the switch points
  - If the majority of your messages are far away, the defaults are probably just fine
- One user obtained 2x speedup in key operation by forcing synchronous protocol
- Later versions of the MPI library will likely support additional switch point variables

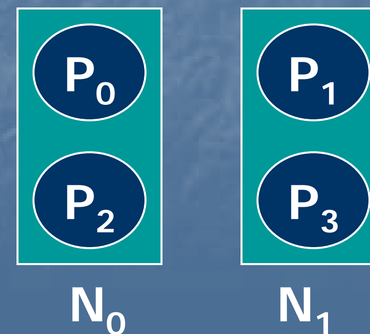
Block vs. Cyclic Distribution

- Srun supports two processes-to-node mappings through '-m' option
  - "-m block" (default)
  - "-m cyclic"

Block



Cyclic



Block vs. Cyclic Distribution

- Depending on how the processes in your application communicate, one mapping may be better than the other
  - Best to group processes that frequently talk to one another close to each other, i.e., on the same node
  - If a process has multiple sets of communication partners, group the set with the costliest communication
- Difficult to generalize, give both settings a try

Contiguous Nodes

- Hardware-based broadcast used when sending to processes residing on nodes physically contiguous wrt switch
  - thunder[40-60] can be done in one Broadcast
  - thunder[40,42-61] requires two – twice as slow
  - thunder[40,42,44-62] requires three …
- Applications which spend much of their time in global calls to Barrier, Broadcast, Reduce, or Allreduce, may notice a difference
  - For these apps, aim to run on largest set of contiguous nodes when you can (e.g., during a DAT)
  - Can exclude fragmented ranges via '-x' srun option
- For apps not dominated by these conditions, fragmented ranges have little impact – don't worry about them

Outline

- Overview
- Programming for Optimal MPI
- Tuning the Runtime Environment
- Wrap-up
  - mpiP – MPI Profiling Library
  - Acknowledgements
  - Resources / Questions

mpiP – MPI Profiling Library

- Answers questions like:
  - How much time is my app spending in MPI?
  - Which MPI call sites are the bottlenecks?
- Easy to use
  - Just replace “srun” with “srun-mpip”
  - No need to rebuild or even relink (in most cases)
- For more info
  - www.llnl.gov/icc/ic/DEG/mpip-llnl.html (LLNL users)
  - www.llnl.gov/CASC/mpip (general users)
  - Developed and maintained by Chris Chambreau

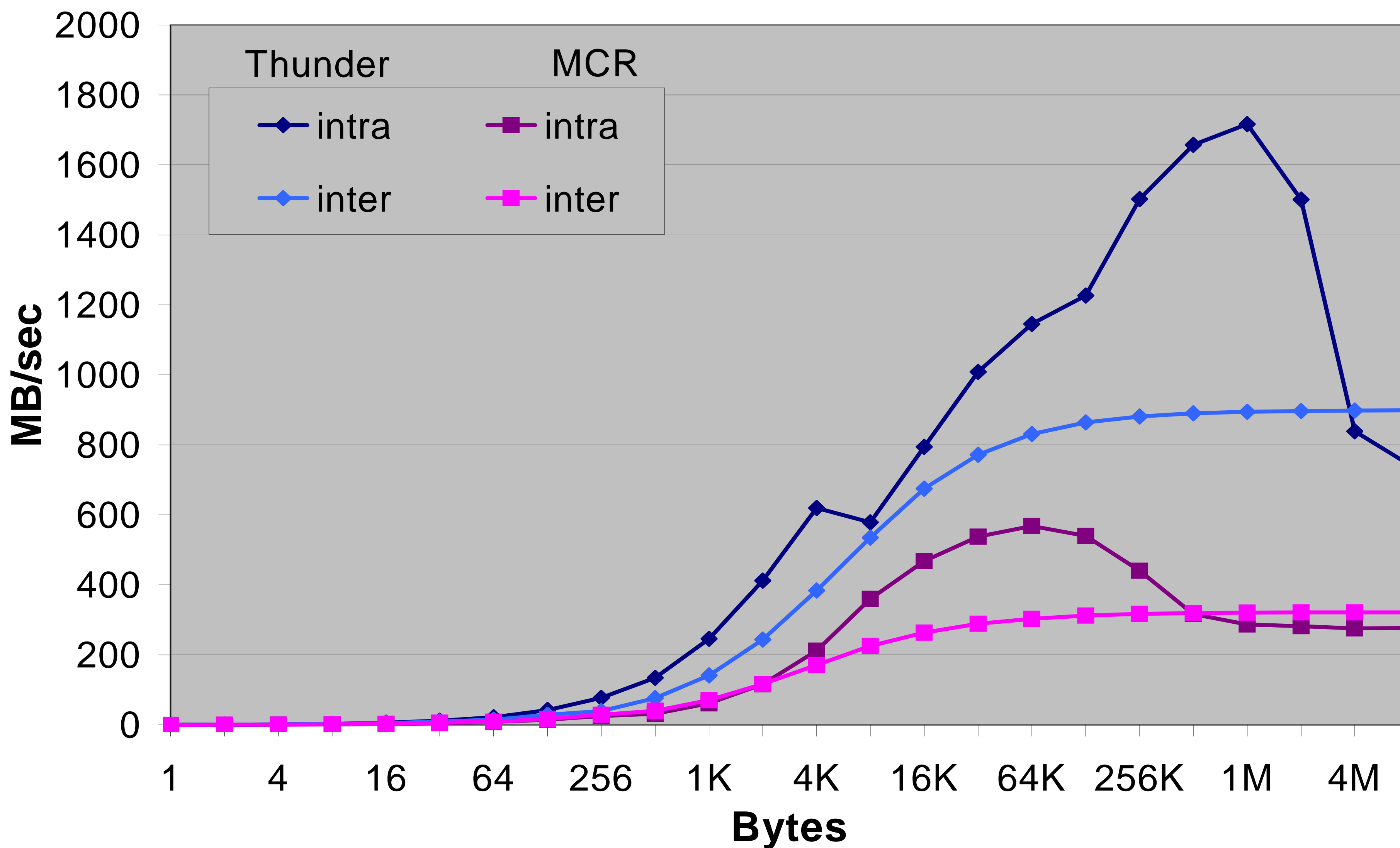
Acknowledgements

- Users and Codes
  - Antonino Ferrante
    - DNSTBL – Direct Numerical Simulation of a Turbulent Boundary Layer
  - John Taylor
    - MM5 – Climate Modeling
  - Alison Kubota
  - Eduardo Bringa
    - MDCASK – Molecular Dynamics
  - Jean-luc Fattebert
    - MGmol – Multi-grid Molecular Dynamics
  - Kathleen McCandless, Mike Zika, Mike Collette
- Quadrics
  - Duncan Roweth and David Addison

Resources / Questions

- LC's MPI web page
  - www.llnl.gov/computing/mpi
  - Links to MPI resources, manuals, general info
    - Performance data
    - MPI environment variables
    - This presentation
- Linux MPI Support
  - Adam Moody moody20@llnl.gov
  - Sheila Faulkner faulkner5@llnl.gov
- Problems
  - LC Hotline

Extra Slides



Effective Bandwidth
Thunder
MCR
intra
inter
intra
inter
MB/sec
0
200
400
600
800
1000
1200
1400
1600
1800
2000
1
4
16
64
256
1K
4K
16K
64K
256K
1M
4M
Bytes