Managing Cray XT MPI Runtime Environment Variables to Optimize and Scale Applications

Geir Johansen

Goals of the Presentation

- Provide users an overview of the implementation of MPI on the Cray XT.
- Explanation of the available Cray XT MPI environment variables.
- Describe scenarios where MPI environment variables are used to optimize or scale an application.
- In short: The information an application analyst should know about Cray XT MPI.

Outline

- Brief description of Message Passing Toolkit 3.0
- Cray XT MPI Environment Variables
  - General
  - Rank Placement
  - SMP
  - Portals
  - Collective Optimizations
  - MPI-IO Hints
- Practical use of MPI Environment Variables

Message Passing Toolkit (MPT) 3.0

- Released on April 24, 2008
- Compute Node Linux (CNL) support, no Catamount support
- MPI based on MPICH2 1.0.4
- Asynchronous release (requires XT OS 2.0.49 or higher)
- Requires use of new asynchronously released compiler drivers
- Same code base used on Cray X2

MPT 3.0

- Uses Process Manager Interface (PMI) to launch MPI processes on the node
  - Interfaces with existing ALPS software (aprun)
  - A PMI daemon process is started on each node
- Support of the SMP device
  - In addition to the current Portals device
  - Optimal messaging path is automatically used.
  - Allows for better optimization for each of the devices
- Improved MPI-IO performance and functionality
- Performance improvements
  - Optimized collectives
  - Latency
  - Others
- Other MPT 3.0 features described throughout presentation

MPI Programming on the Cray XT

- MPI communication was the main focus of Portals design
- Keys to writing optimal MPI code on XT3
  - Pre-post receives of messages.
  - Use non-blocking send/receives
  - Use contiguous data, avoid derived types
- Porting MPI codes to Cray XT3
  - MPI runtime environment variables are used to help optimize and scale the program.

MPI Runtime Environment Variables

- MPI environment variables are documented in the **intro_mpi** man page.
- Buffer sizes can be expressed using ‘K’ (kilobytes) or ‘M’ (megabytes)
- Functions of MPI environment variables
  - Display MPI information
  - Specify optimizations to be used
  - Increase size of message queues and data buffers to scale code
  - Changed the cutoff points where one algorithm is chosen over another

Why are MPI Environment Variables Needed?

- Default settings are set based on the best performance on most codes.
  - Some codes may benefit setting or adjusting environment variable settings.
- Knowledge of application behavior can allow certain optimizations to be turned on.
  - Example: If message sizes are known to be greater than 1 megabyte, then an optimized memcpy can be used that works well for large sizes, but may not work well for smaller sizes.
- Memory space versus communication performance tradeoffs
  - An increase in communication buffer size may increase bandwidth but reduces amount of memory available to the program
- Flow control versus performance tradeoff
  - Using a flow control mechanism may allow application to execute, but at less than optimal performance.
  - User may want to increase size of event queues and buffers and turn off the flow control mechanism

Why are MPI Environment Variables Needed?

- Higher PE counts versus memory usage tradeoff
  - As the application is scaled to higher PE counts the size of the communication event queues and buffers may need to increase. This will reduce amount of memory available to the application.
  - Higher PE counts may require using smaller message sizes.
- User given flexibility to choose cutoff values for collective optimizations
  - The message size value used for determining the use of one collective algorithm over another.
  - The cutoff points for collectives may change for different PE counts

Cray XT MPI Environment Variables

- General
- Rank Placement
- SMP
- Portals
- Collectives
- MPI-IO hints
- *More commonly used environment variables are in italics*

MPI Display Environment Variables

- **MPICH_VERSION_DISPLAY** - displays the version of Cray MPI being used (new to MPT 3.0).
- ***MPICH_ENV_DISPLAY*** - displays MPI environment variables and their values (new to MPT 3.0).
- Helpful to have in the job output when testing different MPI environment variable settings

MPICH_ENV_DISPLAY & MPICH_VERSION_DISPLAY

```
$ aprun -n 2 ./hello
MPI VERSION : CRAY MPICH2 XT version 3.0.0-pre (ANL base 1.0.4p1)
BUILD INFO  : Built Wed Mar 19  5:13:09 2008 (svn rev 6964)
PE 0: MPICH environment settings:
PE 0:   MPICH_ENV_DISPLAY         = 1
PE 0:   MPICH_VERSION_DISPLAY     = 1
PE 0:   MPICH_ABORT_ON_ERROR      = 0
PE 0:   MPICH_CPU_YIELD           = 0
PE 0:   MPICH_RANK_REORDER_METHOD  = 1
PE 0:   MPICH_RANK_REORDER_DISPLAY = 0
PE 0: MPICH/SMP environment settings:
PE 0:   MPICH_SMP_OFF       = 16384
PE 0:   MPICH_MSGS_PER_PROC       = 16384
PE 0:   MPICH_SMPDEV_BUFS_PER_PROC = 32
PE 0:   MPICH_SMP_SINGLE_COPY_SIZE = 131072
PE 0:   MPICH_SMP_SINGLE_COPY_OFF  = 0
PE 0: MPICH/PORTALS environment settings:
PE 0:   MPICH_MAX_SHORT_MSG_SIZE   = 128000
PE 0:   MPICH_UNEX_BUFFER_SIZE     = 62914560
PE 0:   MPICH_PTL_UNEX_EVENTS      = 20480
PE 0:   MPICH_PTL_OTHER_EVENTS     = 2048
PE 0:   MPICH_VSHORT_OFF           = 0
PE 0:   MPICH_MAX_VSHORT_MSG_SIZE  = 1024
PE 0:   MPICH_VSHORT_BUFFERS       = 32
PE 0:   MPICH_PTL_EAGER_LONG       = 0
PE 0:   MPICH_PTL_MATCH_OFF        = 0
PE 0:   MPICH_PTL_SEND_CREDITS     = 0
PE 0: MPICH/COLLECTIVE environment settings:
PE 0:   MPICH_FAST_MEMCPY          = 0
PE 0:   MPICH_COLL_OPT_OFF         = 0
PE 0:   MPICH_BCAST_ONLY_TREE      = 1
PE 0:   MPICH_ALLTOALL_SHORT_MSG   = 1024
PE 0:   MPICH_REDUCE_SHORT_MSG     = 65536
PE 0:   MPICH_ALLREDUCE_LARGE_MSG  = 262144
PE 0:   MPICH_ALLTOALLVW_FCSIZE    = 32
PE 0:   MPICH_ALLTOALLVW_SENDWIN   = 20
PE 0:   MPICH_ALLTOALLVW_RECVWIN   = 20
PE 0: MPICH/MPIIO environment settings:
PE 0:   MPICH_MPIIO_HINTS          = (null)
```

MPI Debug Environment Variables

- ***MPICH_ABORT_ON_ERROR*** – Cause MPICH2 to abort and produce a core dump when an internal MPICH-2 error occurs.
  - In MPT 3.0 performs the function of the MPT 2.0 MPICH_DBMASK=0x200 environment variable

- MPI error messages contain helpful debugging information
  - Many MPT 3.0 messages have been enhanced to suggest MPI environment settings that may resolve the error.

MPICH_CPU_YIELD & PMI_EXIT_QUIET

- **MPICH_CPU_YIELD** – Causes MPI process to call the sched_yield routine to relinquish the processor (new to MPT 3.0).
  - Default is not enabled unless MPI detects oversubscription of processors
  - Useful in cases where a job has oversubscribed the number of CPUs
  - Needed in cases where CPU affinity is set. PathScale compiler enables CPU affinity for OpenMP code.
- **PMI_EXIT_QUIET** – Inhibit PMI from displaying exit info of each PE (New to MPT 3.0).

MPI Rank Reorder Environment Variables

- **MPICH_RANK_REODER_DISPLAY** – displays the node where each MPI rank is executing (New to MPT 3.0)

- ***MPICH_RANK_REORDER_METHOD*** – sets the MPI rank placement scheme. Dual core examples:
  - 0 – Round Robin

NODE	0	1	2	3
RANK	0&4	1&5	2&6	3&7

  - 1 – SMP-style

NODE	0	1	2	3
RANK	0&1	2&3	4&5	6&7

  - 2 – Folded-rank

NODE	0	1	2	3
RANK	0&7	1&6	2&5	3&4

  - 3 – Custom

    Custom rank placement is listed in the file **MPICH_RANK_ORDER**

MPI Rank Reorder Examples (Dual Core)

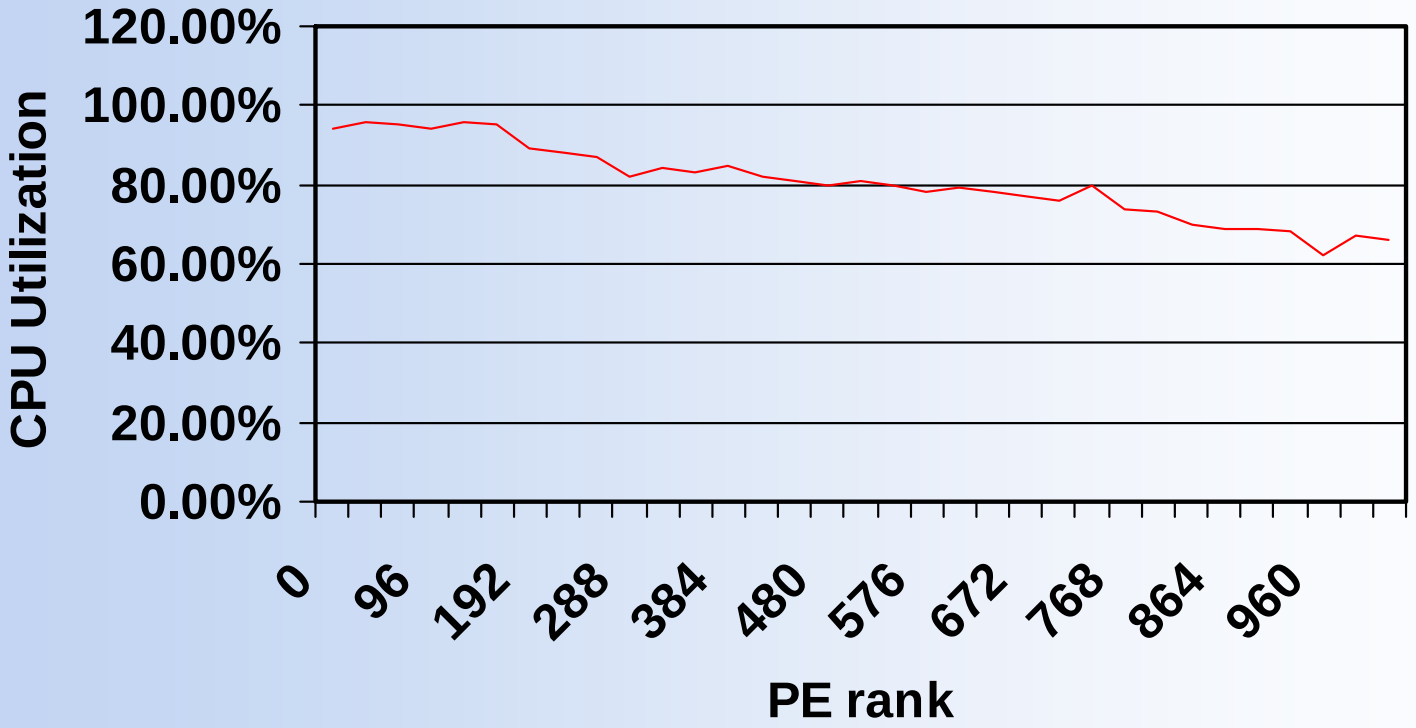
```
$ export MPICH_RANK_REORDER_DISPLAY=1
$ export MPICH_RANK_REORDER_METHOD=0
geir@nid00008:/lus/scratch/geir> aprun -n 8 ./a.out
[PE_0]: MPI rank order: Using round-robin rank ordering
[PE_0]: rank 0 is on nid00469; originally was on nid00469
[PE_0]: rank 1 is on nid00470; originally was on nid00469
[PE_0]: rank 2 is on nid00471; originally was on nid00470
[PE_0]: rank 3 is on nid00473; originally was on nid00470
[PE_0]: rank 4 is on nid00469; originally was on nid00471
[PE_0]: rank 5 is on nid00470; originally was on nid00471
[PE_0]: rank 6 is on nid00471; originally was on nid00473
[PE_0]: rank 7 is on nid00473; originally was on nid00473
Application 280314 resources: utime 0, stime 0
$ export MPICH_RANK_REORDER_METHOD=1
$ aprun -n 8 ./a.out
[PE_0]: MPI rank order: Using default aprun (SMP) rank ordering
[PE_0]: rank 0 is on nid00469; originally was on nid00469
[PE_0]: rank 1 is on nid00469; originally was on nid00469
[PE_0]: rank 2 is on nid00470; originally was on nid00470
[PE_0]: rank 3 is on nid00470; originally was on nid00470
[PE_0]: rank 4 is on nid00471; originally was on nid00471
[PE_0]: rank 5 is on nid00471; originally was on nid00471
[PE_0]: rank 6 is on nid00473; originally was on nid00473
[PE_0]: rank 7 is on nid00473; originally was on nid00473
Application 280321 resources: utime 0, stime 0
$
```

```
$ export MPICH_RANK_REORDER_METHOD=2
$ aprun -n 8 ./a.out
[PE_0]: MPI rank order: Using folded rank ordering.
[PE_0]: rank 0 is on nid00469; originally was on nid00469
[PE_0]: rank 1 is on nid00470; originally was on nid00469
[PE_0]: rank 2 is on nid00471; originally was on nid00470
[PE_0]: rank 3 is on nid00473; originally was on nid00470
[PE_0]: rank 4 is on nid00473; originally was on nid00471
[PE_0]: rank 5 is on nid00471; originally was on nid00471
[PE_0]: rank 6 is on nid00470; originally was on nid00473
[PE_0]: rank 7 is on nid00469; originally was on nid00473
Application 280326 resources: utime 0, stime 0
$ cat >MPICH_RANK_ORDER
7,2,5,0,3,6,1,4
$ export MPICH_RANK_REORDER_METHOD=3
$ aprun -n 8 ./a.out
[PE_0]: MPI rank order: Using a custom rank order from file:
MPICH_RANK_ORDER
[PE_0]: rank 0 is on nid00470; originally was on nid00469
[PE_0]: rank 1 is on nid00473; originally was on nid00469
[PE_0]: rank 2 is on nid00469; originally was on nid00470
[PE_0]: rank 3 is on nid00471; originally was on nid00470
[PE_0]: rank 4 is on nid00473; originally was on nid00471
[PE_0]: rank 5 is on nid00470; originally was on nid00471
[PE_0]: rank 6 is on nid00471; originally was on nid00473
[PE_0]: rank 7 is on nid00469; originally was on nid00473
Application 280461 resources: utime 0, stime 0
```

MPI Rank Placement Environment Variables

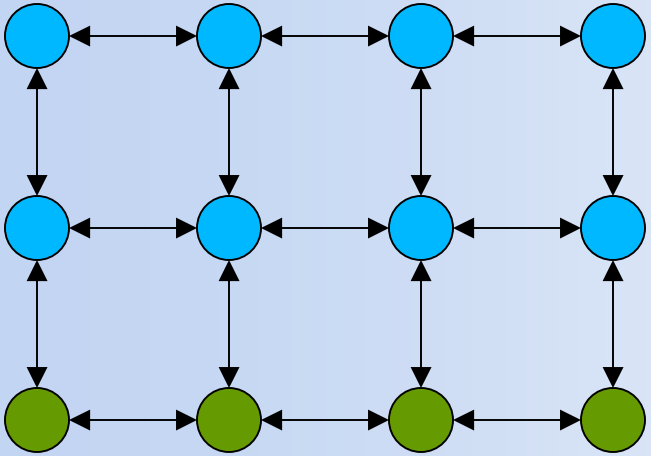
- Default rank placement is determined by the job launcher
  - yod use round robin placement
  - aprun uses SMP placement
- In general, MPI codes perform better using SMP placement
  - Nearest neighbor
- Rank placement can be a very important optimization for some codes. Improvement of load balancing
- Custom rank placement helpful when the decomposition of the data is known

Load Balancing Example

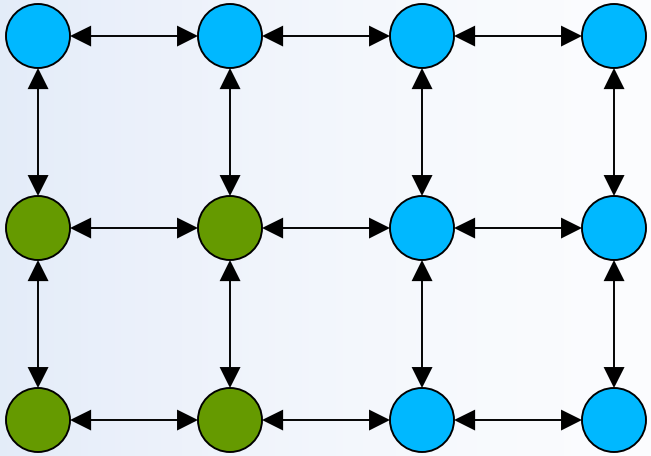


- The higher the PE rank, the lower the CPU utilization. The folded-rank reorder method helped with load balancing and improved the overall performance of the code.

Custom Rank Reorder Example (Quadcore)



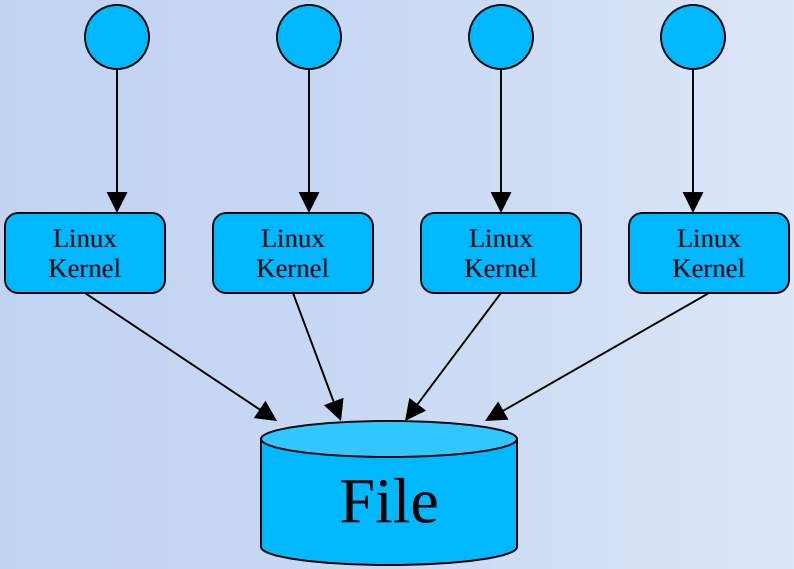
SMP style



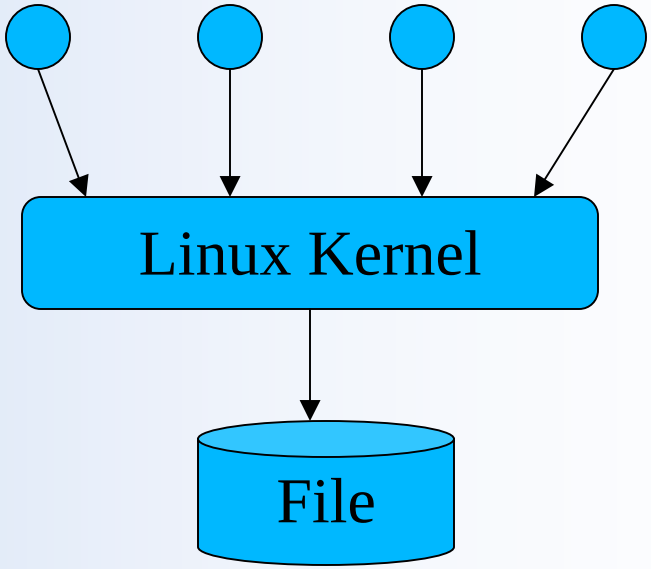
Custom rank reorder

- Using custom rank reorder allowed more communication to be remain on a quadcore node.

Custom Rank Reorder I/O Example (Quadcore)



I/O PEs on different nodes

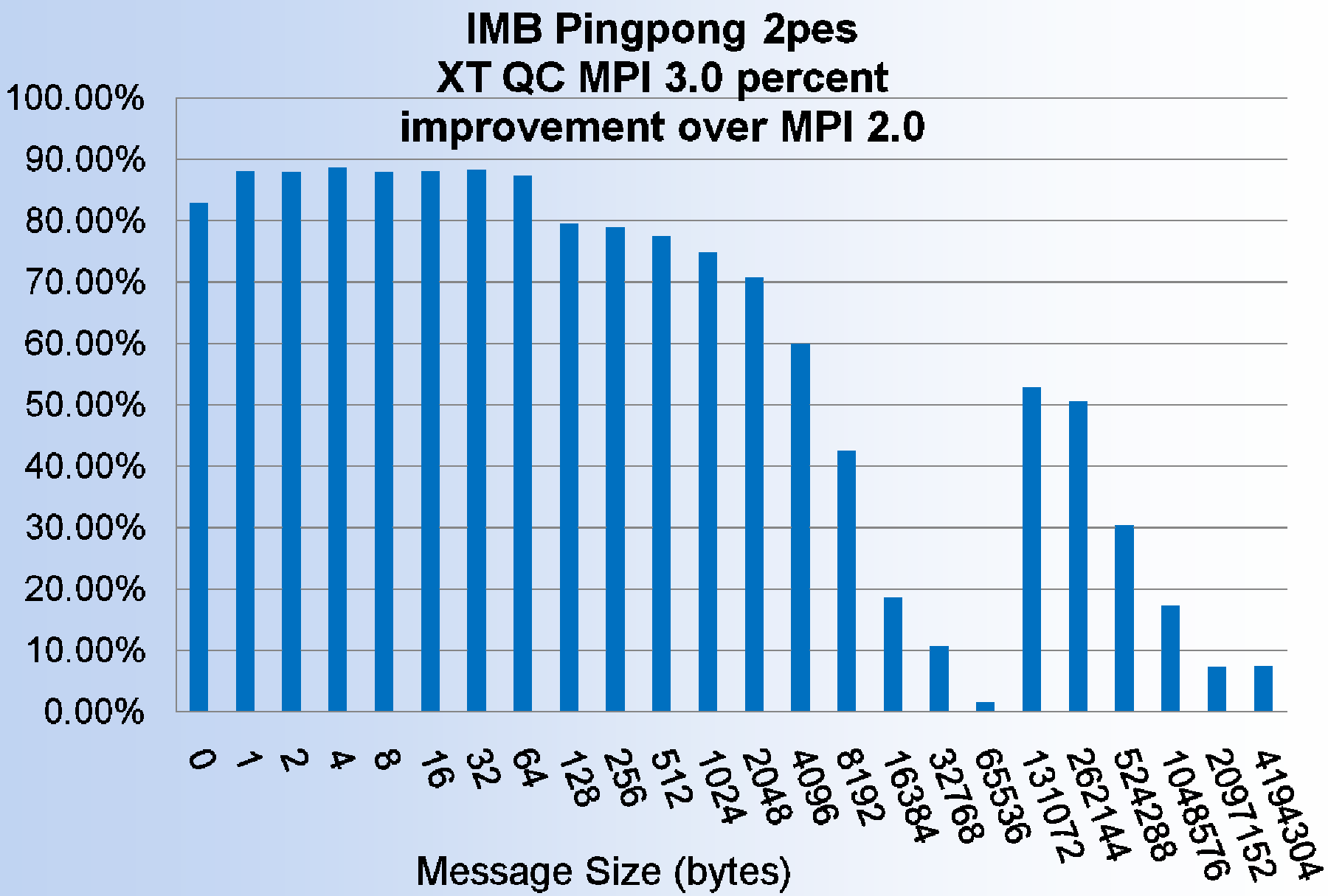


I/O PEs on same node

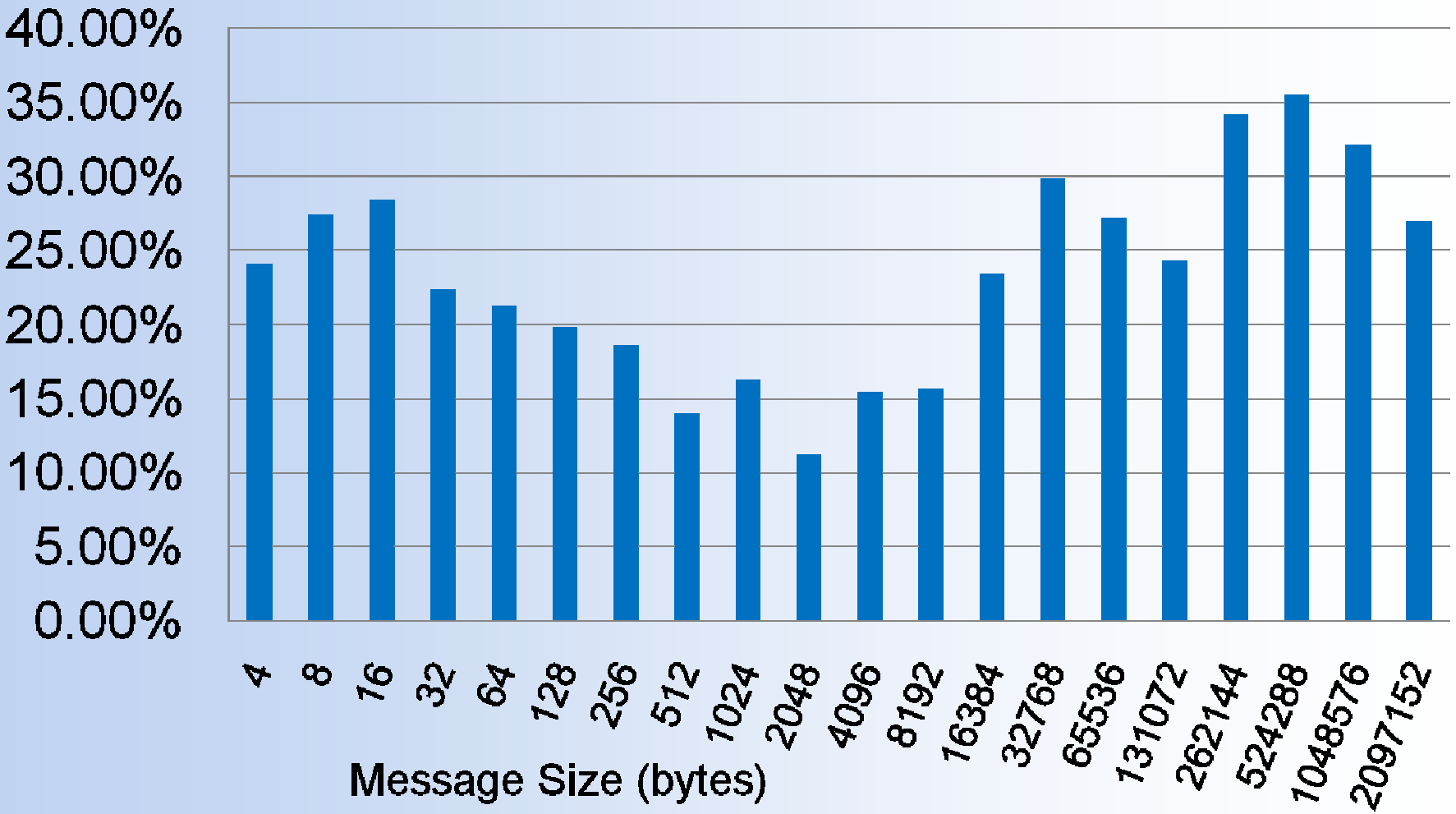
- Assume 4 PEs of a 64 PE job are responsible to writing to a specific file. Better I/O performance if these PEs reside on the same node.

MPI SMP Communication

- SMP ADI (Abstract Device Interface) is new in MPT 3.0
- SMP higher bandwidth, much faster latency than Portals communication
  - Portals performance also enhanced by no longer dealing with intra-node communication.
- Collectives have been optimized to be SMP aware


IMB Pingpong 2pes
XT QC MPI 3.0 percent
improvement over MPI 2.0
100.00%
90.00%
80.00%
70.00%
60.00%
50.00%
40.00%
30.00%
20.00%
10.00%
0.00%
0
1
2
4
8
16
32
64
128
256
512
1024
2048
4096
8192
16384
32768
65536
131072
262144
524288
1048576
2097152
4194304
Message Size (bytes)

IMB Allreduce 256pes
XT QC MPI 3.0 percent improvement over MPI 2.0 with MPI_COLL_OPT_ON



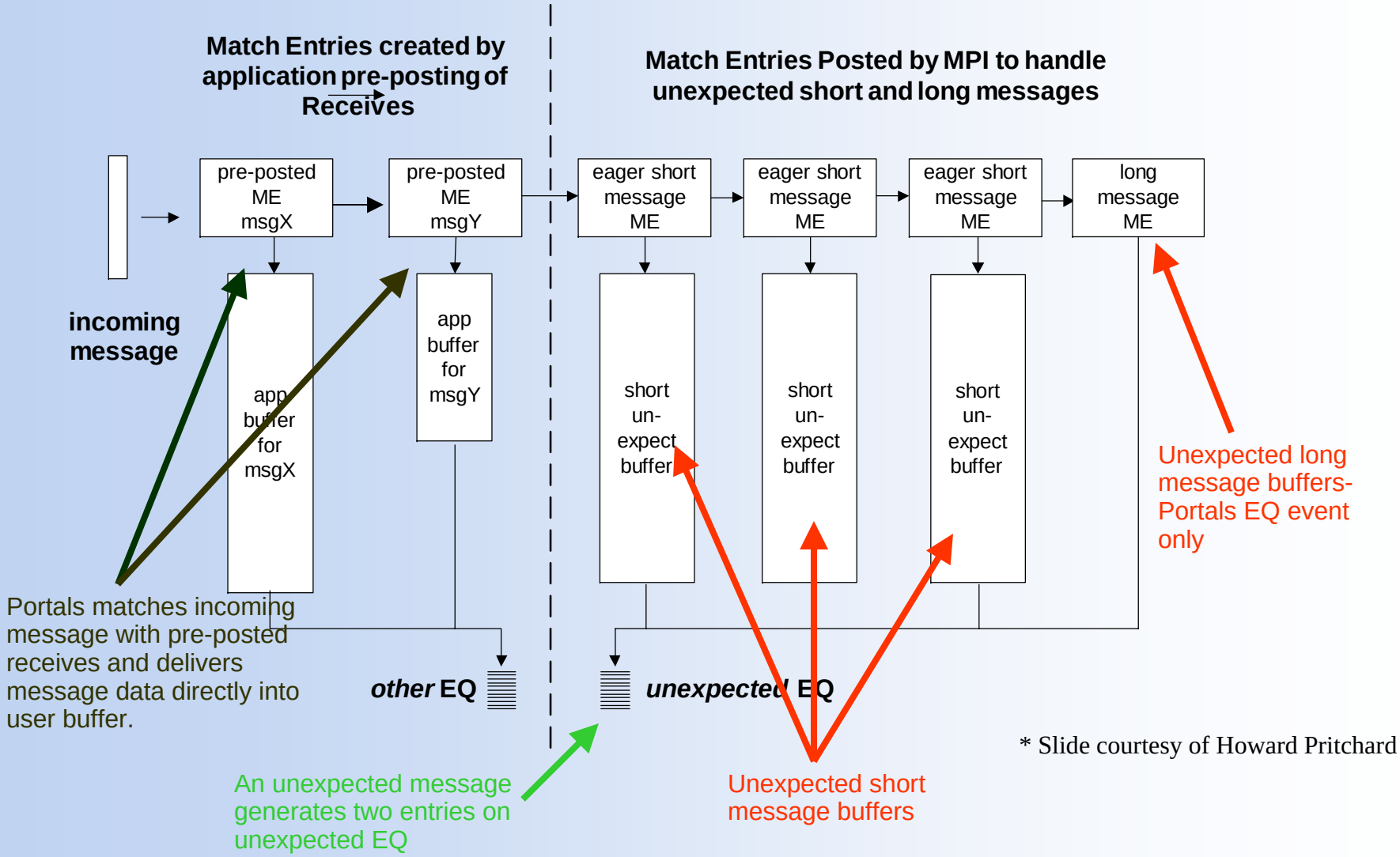
MPI SMP Environment Variables

- **MPICH_SMP_OFF** – Turns off SMP device
  - Used in a rare case where code benefits from using Portals matching instead of MPI matching.
- **MPICH_MSGS_PER_PROCS** – Maximum number of internal MPI message headers.
  - Default is 16384
  - Variable may need to be increased when scaling to higher PE counts
- **MPICH_SMPDEV_BUFS_PER_PROCS** – Number of 16K buffers
  - Default = 32
- **MPICH_SMP_SINGLE_COPY_SIZE** – Minimum message size to use single-copy transfers for on-node messages
  - Default = 128K
  - Single copy requires more system call overhead, so not using single copy is better for smaller buffer size
  - Both point-to-point and collectives performance were considered when choosing default value
- **MPICH_SMP_SINGLE_COPY_OFF** – Disable single-copy mode

Cray XT Portals Communications

- Short Message Eager Protocol
  - Used for messages no more than 128000 bytes (configurable)
  - Sender assumes that receiver has event queue and buffer space to receive message
  - For very short (vshort) messages (<1024), sender copies data to internal buffer before sending PtlPut requests to Portals
- Receiver-pulled Long Message Protocol
  - Default long message protocol
  - Sender sends Portals Header with information for the receiver to get the data
- Eager Long Message Protocol
  - If there is a matching posted receive, Portals delivers message directly into application buffer
  - If no matching receive, receiver gets data like the receiver-pulled method.
  - Eager Long protocol works well only if you can insure that receives are pre-posted!

Cray XT MPI – Receive Side



MPI Portals Environment Variables

- ***MPICH_MAX_SHORT_MSG_SIZE*** – Maximum size of a message to be sent using short eager protocol
  - Default = 128000 bytes
  - Useful optimization technique is to increase this value.
  - Increasing the short message size can require the unexpected buffer size (MPICH_UNEX_BUFFER_SIZE) to be increased.
  - If unexpected buffer size is being overflowed, may need to decrease the max short message size

- ***MPICH_UNEX_BUFFER_SIZE*** – Size of the Portals unexpected buffer
  - Default = 60 Megabytes
  - Buffer size may need to be increased when the number of PEs used is increased
  - If max short buffer size is increased, then the buffer size may need to be increased
  - Increasing buffer size decreases amount of memory available to the application.

MPI Portals Environment Variables

- ***MPICH_PTL_UNEX_EVENTS*** – Number of event queue entries for unexpected messages
  - Default = 20480
  - As numbers of PEs used is increased, then number of event queue entries may need to be increased.
  - Guideline is that value should be at least 2 times the number of PEs being used.

- ***MPICH_PTL_OTHER_EVENTS*** – Number of event queue entries for messages other than unexpected messages
  - Default = 2048
  - In Catamount, MPICH_PTL_UNEX_EVENTS and MPICH_PTL_OTHER_EVENTS came from the same 13K allocation space. This is **not** the case in CNL.

MPI Portals Environment Variables

- **MPICH_VSHORT_OFF** – Disables vshort path optimization
- ***MPICH_MAX_VSHORT_MSG_SIZE*** – Maximum message size to be used for vshort protocol
  - Default = 1024 bytes
  - Maximum value allowed is 16384 bytes
  - Send side optimization
  - Helpful in optimizing codes that frequently send short (<16K) messages
- **MPICH_VSHORT_BUFFERS** – Number of vshort send buffers
  - Default = 32

MPI Portals Environment Variables

- **MPICH_PTL_EAGER_LONG** – Enables eager long protocol
  - Use when receiver pre-posts messages
  - Formerly known as MPICH_PTLS_EAGER_LONG
  - Optimized IMB SendRecv benchmark
- ***MPICH_PTL_MATCH_OFF*** – Disables registration of receive requests with Portal
  - Improves intra-node latency of Portals device (MPT 2.0)
  - Very useful for latency sensitive applications
- ***MPICH_PTL_SEND_CREDITS*** – Enables flow control to prevent the Portals event queue from being overflowed.
  - Value of '-1' should prevent queue overflow in any situation
  - Should only be used as needed, as flow control will result in less optimal performing code.

MPI Portals Error Conditions

- Appendix D of the Cray XT Programming Environment Guide (S-2396) provides a list of common error messages that can occur when exceeding MPI Portals event queues and buffer sizes.
- MPT 3.0 contains improved error messages with suggestions on which environment variables to alter to resolve the issue.

[12] MPICH PtlEQPoll error (PTL_EQ_DROPPED): An event was dropped on the UNEX EQ handle. Try increasing the value of env var MPICH_PTL_UNEX_EVENTS (cur size is 20480).
[241] MPICH has run out of unexpected buffer space.Try increasing the value of env var MPICH_UNEX_BUFFER_SIZE (cur value is 62914560),and/or reducing the size of MPICH_MAX_SHORT_MSG_SIZE (cur value is 128000).
[233] MPICH PtlEQPoll error (PTL_EQ_DROPPED): An event was dropped on the OTHER EQ handle. Try increasing the value of env var MPICH_PTL_OTHER_EVENTS (cur size is 2048).

Scaling MPI_Alltoall to 30,000 nodes

- Total memory required just for alltoall communication:

  2 * 30000 * message-size

  Used a message size of 16K, as this will use almost 1GB of memory (running on a machine with 2G per CPU)

- The number of unexpected event queue entries should be at least 2 times the number of PEs. Environment variable **MPICH_PTL_UNEX_EVENTS** was set to 70000.

- The size of the Portals unexpected buffer needed to be increased. Setting **MPICH_UNEX_BUFFER_SIZE**=100M was found to be sufficient

* IMB-MPI Alltoall executed on 30,000 nodes at ORNL by Jeff Becklehimer and Kim McMahon

MPI Collective Environment Variables

- ***MPICH_FAST_MEMCPY*** – Use optimized memcpy routine
  - Benefits occur with messages 256K and large
  - Helpful for many real world applications
  - Not turned on by default

- **MPICH_COLL_OPT_OFF** – Disables collective optimizations
  - Affects MPI_Allreduce and MPI_Barrier
  - In MPT 2.0, needed to turn on collective optimizations using MPICH_COLL_OPT_ON

- **MPICH_BCAST_ONLY_TREE** – Setting to ‘0’ enables the ring algorithm in MPI_Bcast for nonpower of two sizes.
  - Default = 1
  - Rarely used

MPI Collective Environment Variables

- **MPICH_ALLTOALL_SHORT_MSG** – Cut-off point for using the store and forward Alltoall algorithm
  - Default = 1024 (Default in MPT 2.0 is 512)
  - Store and forward algorithm very effective for small message sizes and large PE counts
- **MPICH_REDUCE_SHORT_MSG** – Cut-off point for using binomial tree algorithm (smaller values) versus a reduce-scatter algorithm (larger value).
  - Default = 65536
- **MPICH_ALLREDUCE_LARGE_MSG** – Cutoff point for the SMP-aware MPI_allreduce algorithm.
  - Default = 262144
  - Larger message sizes use default MPICH2 algorithm
  - New in MPT 3.0

MPI Collective Environment Variables

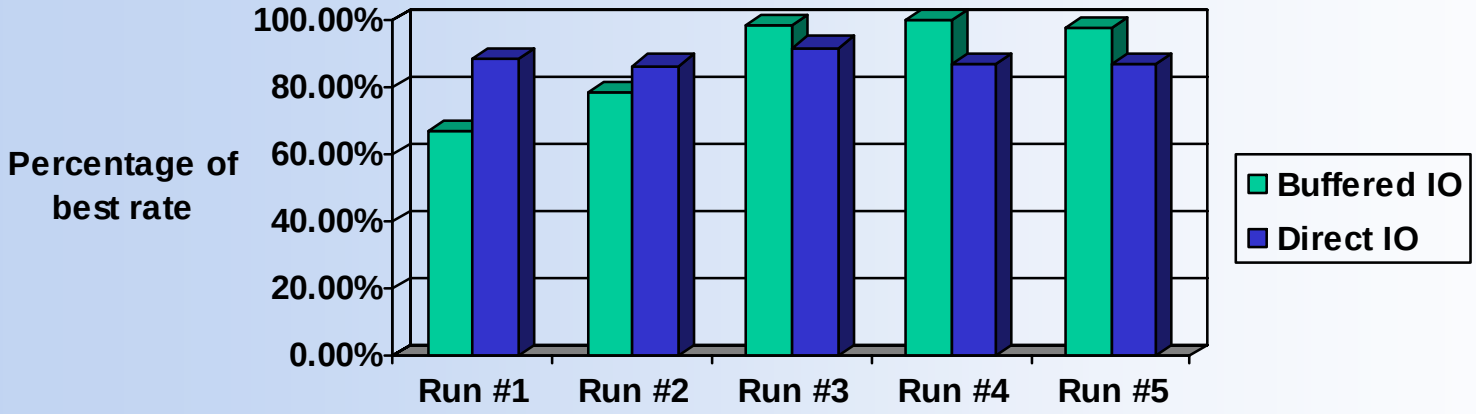
- **MPICH_ALLTOALLVW_FCSIZE** – The cutoff size of the number of communicators when the flow-control ("FC") MPI_Alltoall[vw] algorithm is used
  - Default = 32 (Default in MPT 2.0 was 120)
  - Flow control is used to prevent Seastar CAM entries from overflowing
  - Default values were changed from MPT 2.0, as they were shown to perform better
- **MPICH_ALLTOALLVW_RECVWIN** – The number of concurrent Irecv operations allowed when the flow-control MPI_Alltoall[vw] algorithm is used
  - Default = 20 (Default in MPT 2.0 was 80)
- **MPICH_ALLTOALLVW_SENDWIN** – The number of concurrent Isend operations allowed when the flow-control MPI_Alltoall[vw] algorithm is used
  - Default = 20 (Default in MPT 2.0 was 100)

MPI MPI-IO Hints

- **MPICH_MPIIO_HINTS_DISPLAY** – Rank 0 displays the name and values of the MPI-IO hints (New in MPT 3.0)
- **MPICH_MPIO_HINTS** – Sets the MPI-IO hints for files opened with the MPI_File_Open routine (New in MPT 3.0).
  - Overrides any values set in the application by the MPI_Info_set routine
  - Enabling collective buffering on writes (romio_cb_write) and disabling data sieving on writes (romio_ds_write) have been used by Cray XT MPI codes to improve I/O performance
  - Following hints supported:

direct_io	cb_nodes	romio_ds_write
romio_cb_read	cb_config_list	ind_rd_buffer_size
romio_cb_write	romio_no_indep_rw	ind_wr_buffer_size
cb_buffer_size	romio_ds_read	

MPI-IO HINTS direct_io Example



- Rates for direct I/O were more consistent
  - - Buffered I/O rate is dependent on other kernel activities
  - - Buffer I/O sometimes out performed direct I/O

- MPIIO direct_io hint will only perform direct I/O when call with aligned data, otherwise it will use buffered I/O

* MPI-IO tests executed by David Knaak and Dick Sandness

MPI Env Variables to Optimize Performance

- Try the different rank reorder methods
  - Easy to switch the between round-robin, SMP, and folded-rank methods, may result in significant optimization
  - The optimal rank reorder method not always obvious, communication distance versus load balancing tradeoffs
  - If using Catamount, you will want to at least try SMP style
- If the code is **latency sensitive**, set the MPICH_PTL_MATCH_OFF option
  - Example is LS-DYNA
- If using Catamount, try setting **MPICH_COLL_OPT_ON** environment variable.

MPI Env Variables to Optimize Performance

- If **size of messages are less than 16K**
  - Set the MPICH_MAX_VSHORT_MSG_SIZE to 16382
- If **message sizes are greater than 128000 bytes**
  - Increase MPICH_MAX_SHORT_MSG_SIZE to allow more messages to be sent with the Portals Short Eager protocol
  - Increasing this environment variable will likely require an increase of the unexpected buffer (MPICH_UNEX_BUFFER_SIZE) and possibly the unexpected event queue (MPICH_PTL_UNEX_EVENTS).
- If **message sizes are greater than 256K bytes**
  - Use the MPICH_FAST_MEMCPY environment variable!
  - For messages less than 256K the performance is about the same as the default. Only degradation is seen in benchmark codes (possible caching effect)

MPI Env Variables to Optimize Performance

- If running at higher PE counts (>1024), then user may want to experiment with the cutoff values for MPI_Alltoall, and MPI_Reduce.

MPI Env Variables to Scale a Code

- The number of Portals unexpected event queue entries (**MPICH_PTL_UNEX_EVENTS**) should be at least two times the number of PEs being used
- An increase in the number of unexpected event queue entries (**MPICH_PTL_UNEX_EVENTS**) may also likely require an increase to the size of the Portals unexpected buffer (**MPICH_UNEX_BUFFER_SIZE**)
- If the Portals unexpected buffer size can not be increased enough to handle application, then the message size that uses the short message eager protocol may need to be reduced (**MPICH_MAX_SHORT_MSG_SIZE**).
  - Forces some messages to use the long message receiver-pulled protocol
  - Alternatively, the code could be altered to use smaller message sizes

MPI Env Variables to Scale a Code

- If the Portals unexpected event queue can not be increased enough, then flow control may need to be enabled. Setting **MPICH_PTL_SEND_CREDITS** to '-1' will invoke a flow control algorithm to prevent event queue from overflowing.
- The number of Portals other event queue entries (**MPICH_PTL_OTHER_EVENTS**) may need to be increased when the PE count is increased.
- The number of MPICH messages (**MPICH_MSGS_PER_PROC**) may possibly need to be increased when the PE count is increased.

Cray XT Future Enhancements

- Additional SMP aware collective optimizations (Alltoall, possibly Bcast).
- More MPI-IO optimizations
- Support for MPI to run over 100,000 processors

Questions

- geir@cray.com