

How to Make Best Use of Cray MPI on the XT

Outline

- Overview of Cray Message Passing Toolkit (MPT)
- Key Cray MPI Environment Variables
- Cray MPI Collectives
- Cray MPI Point-to-Point Messaging Techniques
- Memory Usage when Scaling Applications
- When Something Goes Wrong - Cray MPI Error Messages

Cray Message Passing Toolkit (MPT) 3.x+4.x

■ Toolkit includes MPI and SHMEM

- ✿ MPI 3 based of MPICH2 version 1.0.6 from ANL
- ✿ MPI 4 based of MPICH2 version 1.1.1p1 from ANL
- ✿ Support for multiple compilers (CCE, PGI, Pathscale, GNU, INTEL)
- ✿ Numerous Cray enhancements and optimizations

■ What Unique Features does CRAY MPI provide for XT

- ✿ Custom Portals device driver
- ✿ Custom Shared Memory (SMP) device driver
- ✿ Multi-device implementation for a single job
 - ▶ Optimal messaging path is selected automatically
- ✿ Optimized Collectives
- ✿ MPI I/O Enhancements
- ✿ Support for up to 256,000 MPI ranks
- ✿ Custom Process Manager Interface (PMI) for launching
 - ▶ Interfaces with existing ALPs software (aprun)
 - ▶ A PMI daemon process is started on each node
 - ▶ Support for Process-to-CPU affinity
 - ▶ Support for Rank Re-Ordering

Key Cray MPI Environment Variables

■ Why use MPI environment variables?

- ⚙️ Allow users to tweak optimizations for specific application behavior
- ⚙️ Flexibility to choose cutoff values for collective optimizations
- ⚙️ Determine maximum size of internal MPI buffers/queues

■ MPI Display Variables

⚙️ **MPICH_VERSION_DISPLAY**

- ▶ Displays version of Cray MPI being used
- ▶ strings `./mpi.exe | grep VERSION`

MPI VERSION : CRAY MPICH2 XT version 3.1.2 (ANL base 1.0.6)

BUILD INFO : Built Mon Feb 16 10:20:17 2009 (svn rev 7304)

⚙️ **MPICH_ENV_DISPLAY**

- ▶ Displays all MPI env variables and their current values
- ▶ Helpful to determine what defaults are set to

MPICH_ENV_DISPLAY & MPICH_VERSION_DISPLAY

MPI VERSION : CRAY MPICH2 XT version 3.1.2-pre (ANL base 1.0.6)

BUILD INFO : Built Thu Feb 26 3:58:36 2009 (svn rev 7308)

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_MAX_THREAD_SAFETY     = single
PE 0:  MPICH_MSGS_PER_PROC         = 16384
PE 0:  MPICH/SMP environment settings:
PE 0:  MPICH_SMP_OFF               = 0
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_COLL_SYNC            = 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_REDUCE_LARGE_MSG     = 131072
PE 0:  MPICH_ALLREDUCE_LARGE_MSG  = 262144
PE 0:  MPICH_ALLGATHER_VSHORT_MSG = 2048
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_DISPLAY  = 0
PE 0:  MPICH_MPIIO_CB_ALIGN       = 0
PE 0:  MPICH_MPIIO_HINTS          = NULL
```

Auto-Scaling MPI Environment Variables

- Key MPI variables that **change** their default values dependent on job size

MPICH_MAX_SHORT_MSG_SIZE	MPICH_PTL_UNEX_EVENTS
MPICH_UNEX_BUFFER_SIZE	MPICH_PTL_OTHER_EVENTS

- ✿ New since MPT 3.1
 - ✿ Aids in scaling applications
 - ✿ “Default” values are based on total number of ranks in job
 - ✿ See MPI man page for specific formulas used
- We don't always get it right
 - ✿ Adjusted defaults aren't perfect for all applications
 - ✿ Assumes a somewhat communication-balanced application
 - ✿ Users can always override the new defaults

Cray MPI XT Portals Communications

■ Short Message **Eager** Protocol

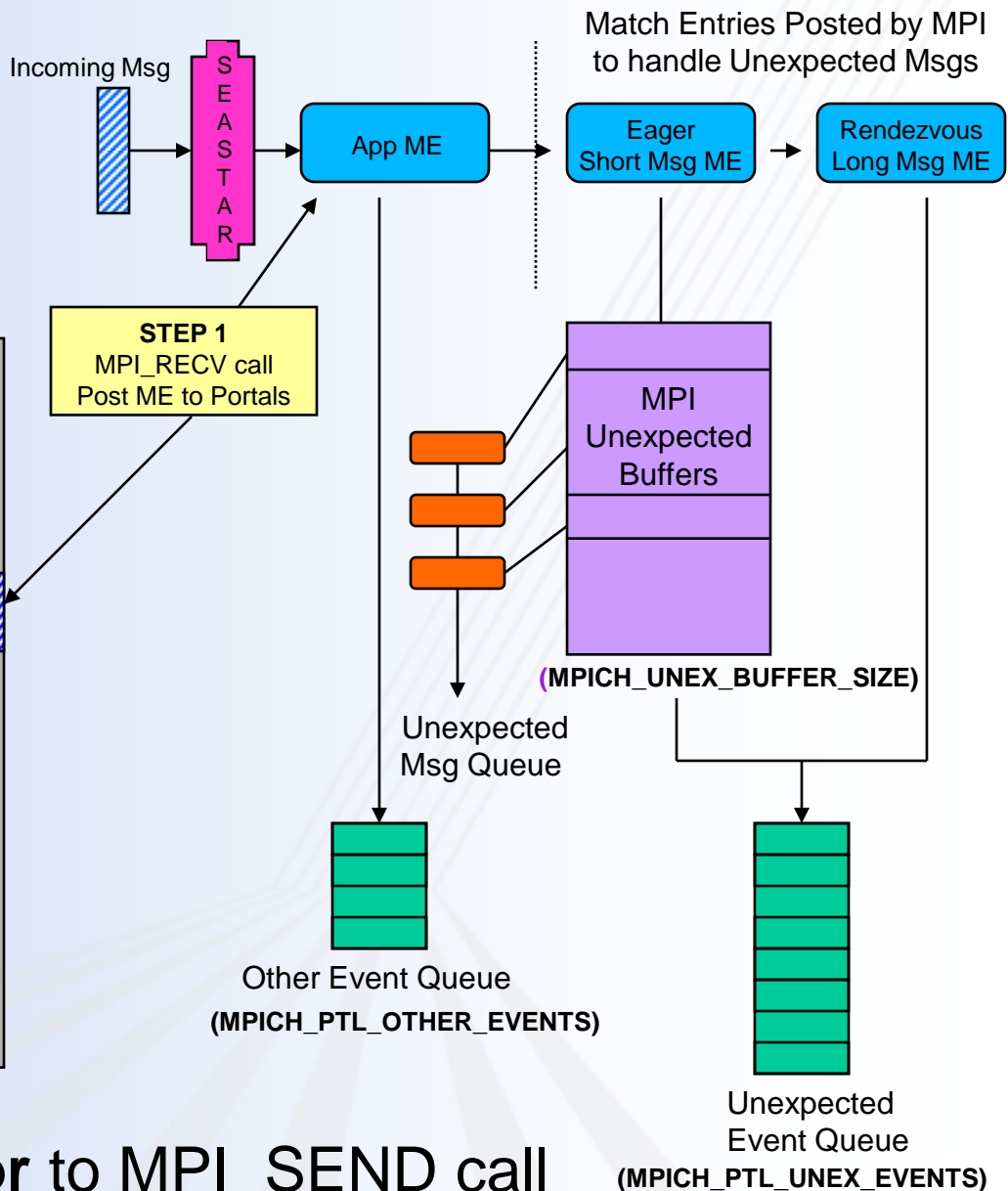
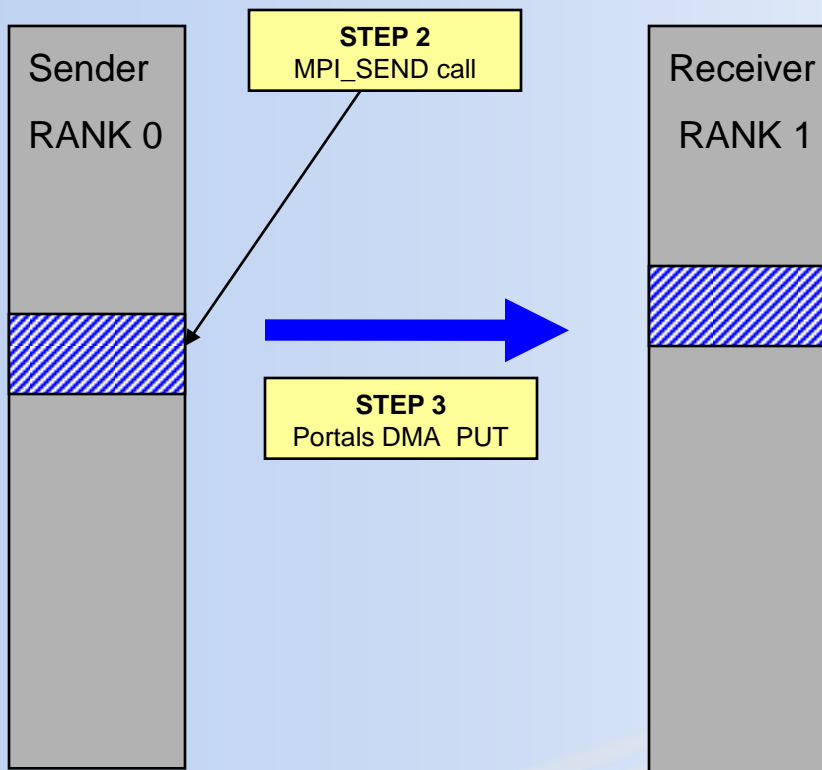
- ✿ The sending rank “pushes” the message to the receiving rank
- ✿ Used for messages **MPICH_MAX_SHORT_MSG_SIZE** bytes or less
- ✿ Sender assumes that receiver can handle the message
 - ▶ Matching receive is posted - or -
 - ▶ Has available event queue entries (**MPICH_PTL_UNEX_EVENTS**) and buffer space (**MPICH_UNEX_BUFFER_SIZE**) to store the message

■ Long Message **Rendezvous** Protocol

- ✿ Messages are “pulled” by the receiving rank
- ✿ Used for messages greater than **MPICH_MAX_SHORT_MSG_SIZE** bytes
- ✿ Sender sends MPI Header with information for the receiver to pull over the data
- ✿ Data is sent only after matching receive is posted by receiving rank

MPT Eager Protocol

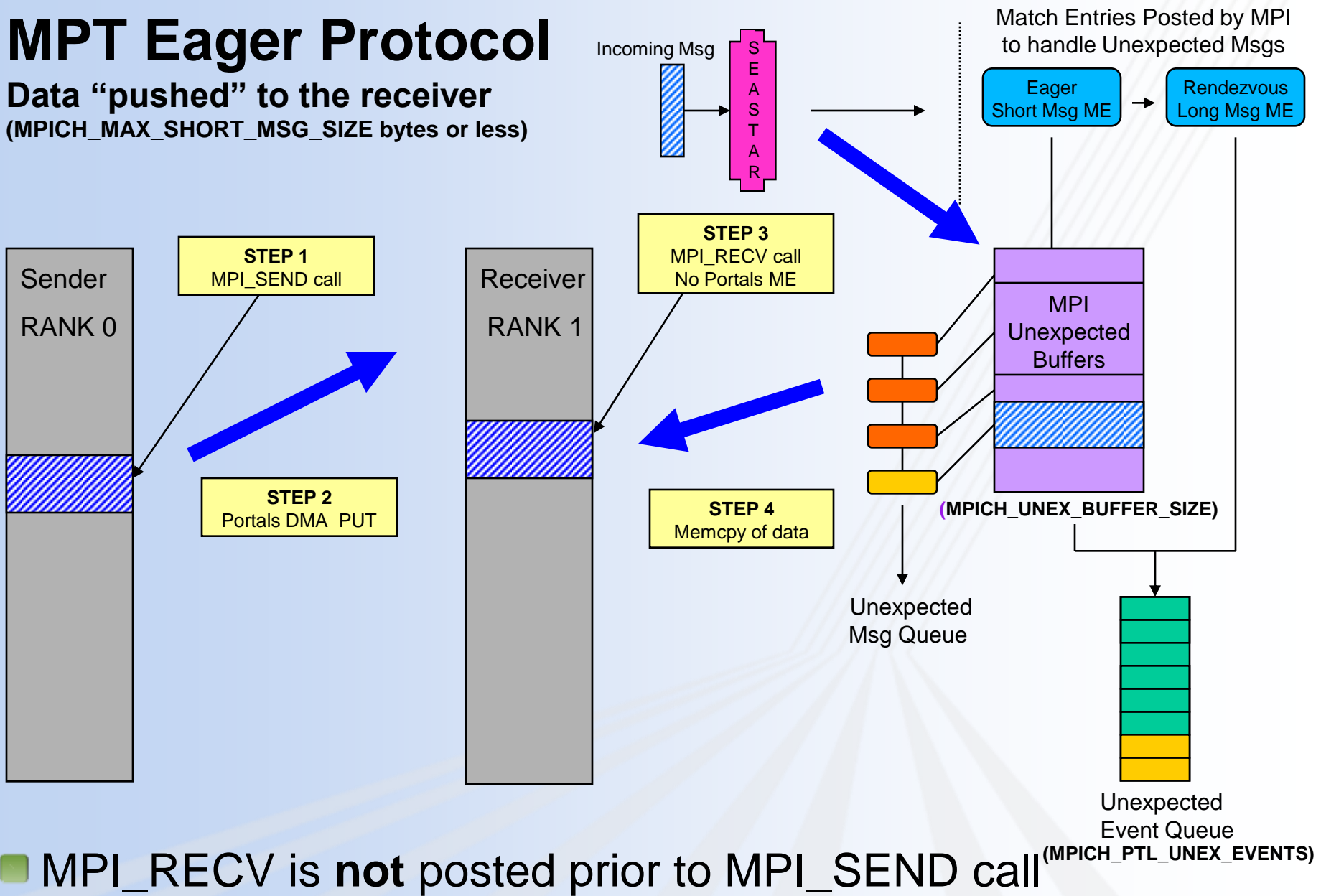
Data "pushed" to the receiver
(MPICH_MAX_SHORT_MSG_SIZE bytes or less)



■ MPI_RECV is posted **prior** to MPI_SEND call

MPT Eager Protocol

Data "pushed" to the receiver
(MPICH_MAX_SHORT_MSG_SIZE bytes or less)

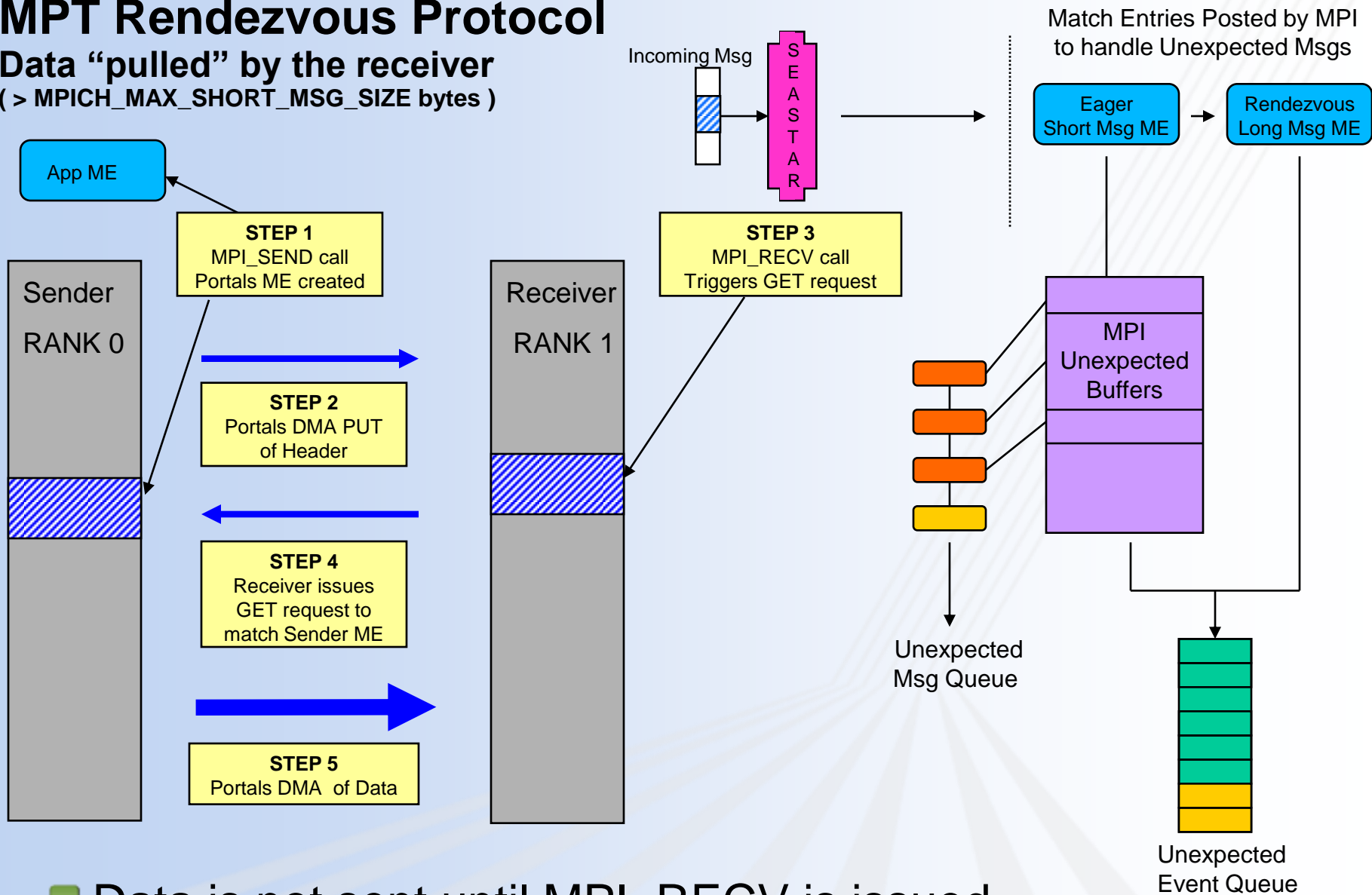


■ MPI_RECV is **not** posted prior to MPI_SEND call

MPT Rendezvous Protocol

Data "pulled" by the receiver

(> MPICH_MAX_SHORT_MSG_SIZE bytes)



■ Data is not sent until MPI_RECV is issued

Auto-Scaling MPI Environment Variables

- Default values for various MPI jobs sizes

MPI Environment Variable Name	1,000 PEs	10,000 PEs	50,000 PEs	100,000 PEs
MPICH_MAX_SHORT_MSG_SIZE (This size determines whether the message uses the Eager or Rendezvous protocol)	128,000 bytes	20,480	4096	2048
MPICH_UNEX_BUFFER_SIZE (The buffer allocated to hold the unexpected Eager data)	60 MB	60 MB	150 MB	260 MB
MPICH_PTL_UNEX_EVENTS (Portals generates <u>two</u> events for each unexpected message received)	20,480 events	22,000	110,000	220,000
MPICH_PTL_OTHER_EVENTS (Portals send-side and expected events)	2048 events	2500	12,500	25,000

Cray MPI Collectives

■ Cray Optimized Collectives

- ✿ Work for any intra-communicator (not just MPI_COMM_WORLD)
- ✿ Enabled by default
- ✿ Many have user-adjustable cross-over points (see man page)
- ✿ Can be selectively disabled via MPICH_COLL_OPT_OFF
 - ▶ `export MPICH_COLL_OPT_OFF=mpi_bcast,mpi_allgather`

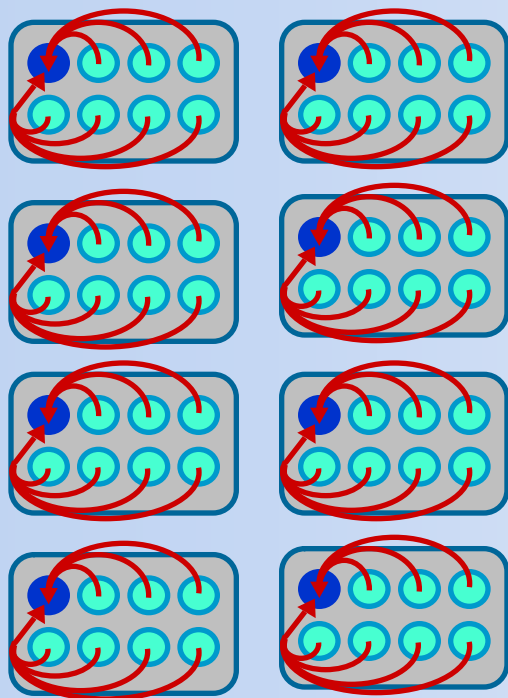
■ Cray MPI_Alltoallv / MPI_Alltoallw algorithm

- ✿ Pairwise exchange with windows
- ✿ Default window sizes set to allow 20 simultaneous sends/recvs
- ✿ Set window sizes to 1 when scaling with medium/large messages
 - ▶ `export MPICH_ALLTOALLVW_SENDWIN=1`
 - ▶ `export MPICH_ALLTOALLVW_RECVWIN=1`

■ Cray-Optimized SMP-aware Collectives

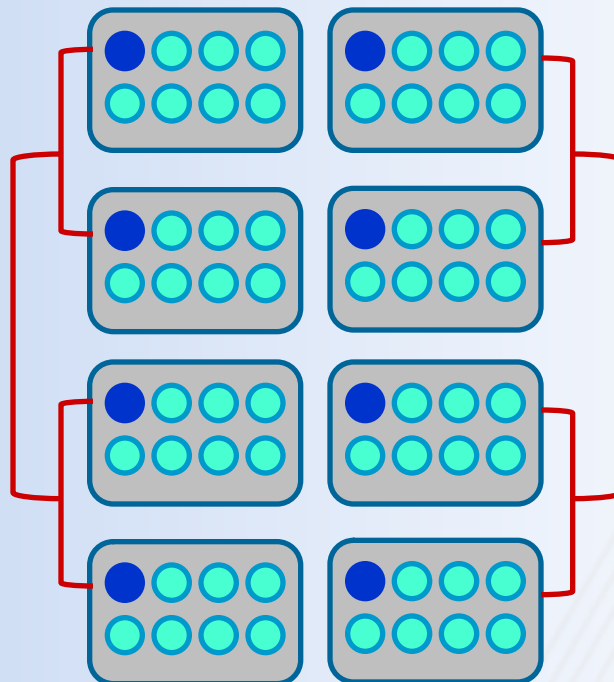
- ✿ MPI_Allreduce
- ✿ MPI_Barrier
- ✿ MPI_Bcast (new in MPT 3.1.1)
- ✿ MPI_Reduce (new in MPT 3.1.2)

SMP-aware Collectives – Allreduce Example



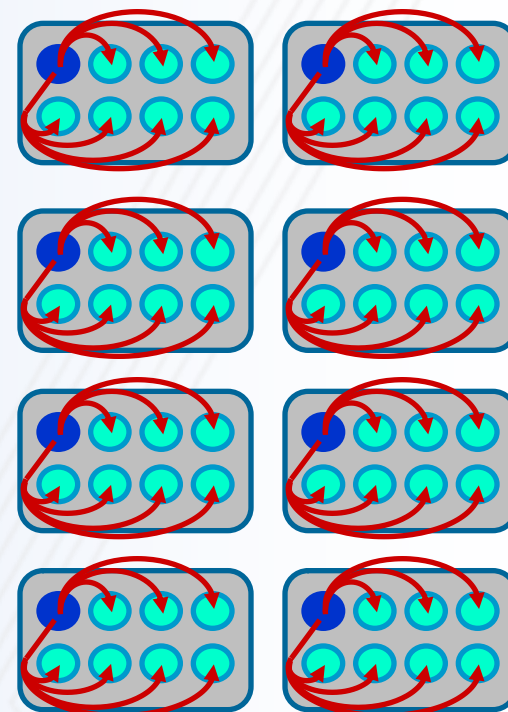
STEP 1

Identify Node-Captain rank.
Perform a local on-node
reduction to node-captain.
NO network traffic.



STEP 2

Perform an Allreduce with node-
captains only. This reduces the
process count by a factor of 8 on
XT5.



STEP 3

Perform a local on-node
bcast. NO network traffic.

Cray MPI Point-to-Point Messaging

- Pre-posting receives is generally a good idea
 - ✿ For EAGER messages, this avoids an extra memcpy
 - ✿ Portals/Seastar handles the data copy directly into the user buffer
 - ✿ Can off-load work from CPU
 - ✿ Avoid posting thousands of receives

- Non-contiguous data types
 - ✿ More efficient to use contiguous data types for message transfers
 - ✿ If discontinuous, MPI must:
 - ▶ Send side: Allocate temp buffer, pack user data into temp buffer
 - ▶ Entire message is sent over network as contiguous
 - ▶ Recv side: Unpack temp buffer into user buffer

- Avoid “swamping” a busy rank with thousands of messages
 - ✿ Reduce `MPICH_MAX_SHORT_MSG_SIZE` to force rendezvous protocol
 - ✿ Consider enabling `MPICH_PTL_SEND_CREDITS` “flow-control” feature
 - ✿ Modify code to use explicit handshaking to minimize number of in-flight messages

Memory Usage when Scaling Applications

- Watch Memory Footprint as Applications Scale
 - ✿ Understand application memory usage as process count increases
 - ✿ MPI unexpected buffers the largest consumer for MPI internally
 - ▶ Default is 260MB per process for 150,000 rank job
 - ▶ Decrease by reducing size of `MPICH_UNEX_BUFFER_SIZE`

- MPI Collective Memory Usage
 - ✿ When scaling, watch use of collectives that accumulate data on a per-rank basis
 - ✿ `MPI_Alltoall`, `MPI_Allgather`, `MPI_Gather`, etc.

- Options to Decrease Memory Footprint
 - ✿ Decrease process density per node (`-N8` vs `-N6`, `-N4`, `-N2`, `-N1`)
 - ▶ Specify `aprun` options to use both NUMA nodes on a socket
 - ✿ Consider hybrid MPI + OMP approach

Memory Usage for MPI_Alltoall

- Alltoall function requires sendbuf and recvbuf parameters
 - ✿ Each rank needs to allocate:
(count * sizeof(datatype) * num_ranks) bytes for each buffer
 - ✿ This adds up quickly when scaling to extreme process counts!

Consider the following code snippet...

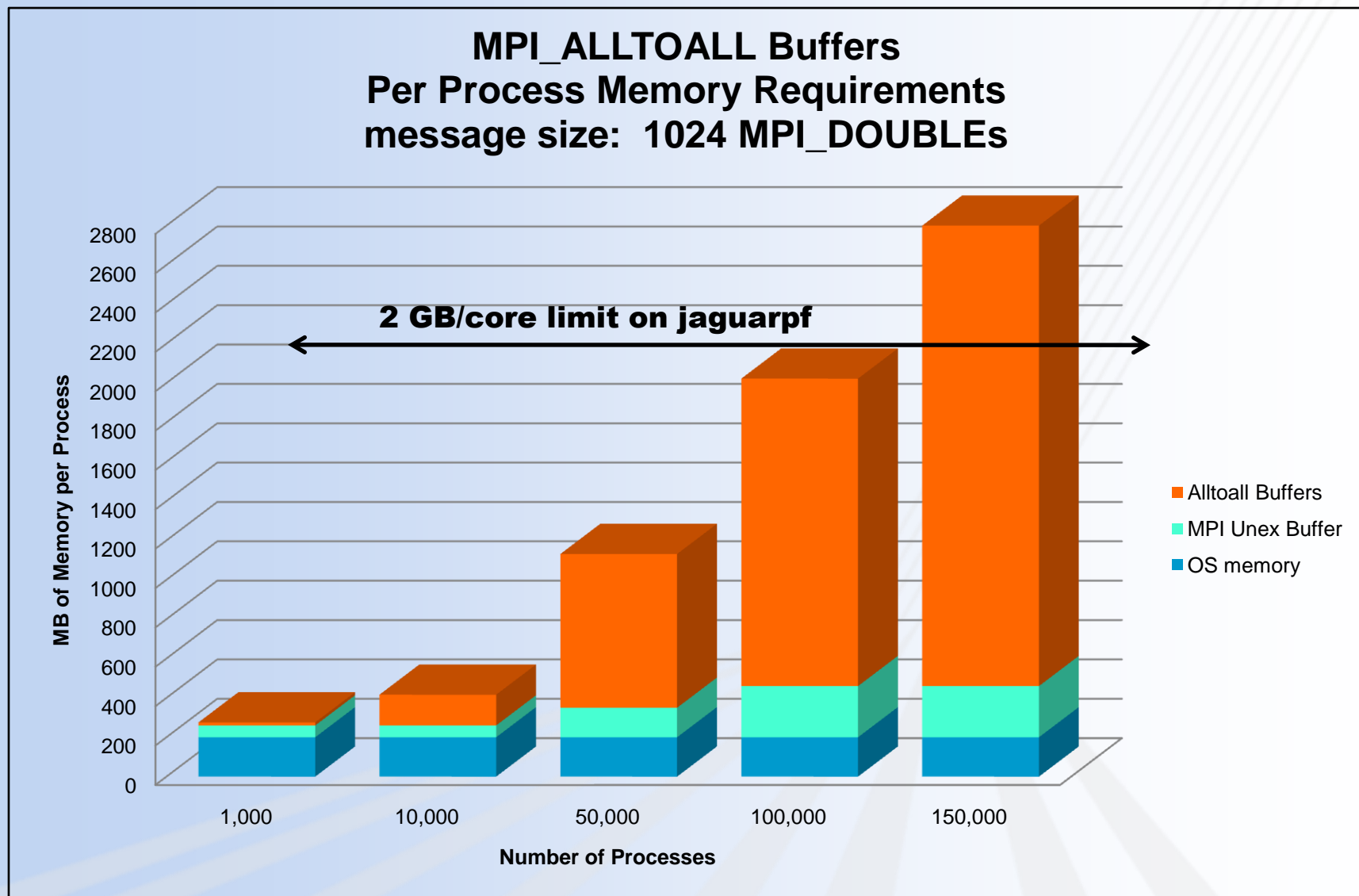
```
MPI_Comm_rank( MPI_COMM_WORLD, &rank );
MPI_Comm_size( MPI_COMM_WORLD, &size );

count    = 1024;
sendbuf  = (double *) malloc(count * sizeof(double) * size);
recvbuf  = (double *) malloc(count * sizeof(double) * size);

...

MPI_Alltoall(sendbuf, count, MPI_DOUBLE, recvbuf,
             count, MPI_DOUBLE, MPI_COMM_WORLD);
```

Memory Usage for Scaling MPI_Alltoall



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 the code is **latency sensitive**, set the `MPICH_PTL_MATCH_OFF` option

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`.

When Something Goes Wrong - MPI Error Messages

- If a rank exits abnormally, PMI daemon reports the error

```
_pmi_daemon(SIGCHLD): PE 1036 exit signal Segmentation fault
_pmi_daemon(SIGCHLD): PE 0 exit signal Killed
_pmi_daemon(SIGCHLD): PE 1 exit signal Killed
_pmi_daemon(SIGCHLD): PE 2 exit signal Killed
...
_pmi_daemon(SIGCHLD): PE 1035 exit signal Killed
```

- To quiet the PMI daemon, use: `export PMI_QUIET=1`
- Rely on single aprun error message for clues

```
[NID 3343]Apid 250839: initiated application termination
Application 250839 exit codes: 139
Application 250839 exit signals: Killed
Application 250839 resources: utime 0, stime 0
```

Subtract 128 from aprun exit code to get the fatal signal number. In this case, signal 11 is a segmentation fault. See aprun man page for more info.

When Something Goes Wrong - MPI Error Messages

- For fatal signals or MPICH errors, get a corefile/traceback
 - ✿ Unlimit coredumpsize limit
 - ✿ export MPICH_ABORT_ON_ERROR=1
 - ✿ One corefile is produced by first rank to hit the problem

```
Fatal error in MPI_Wait: Invalid MPI_Request, error stack:  
MPI_Wait(156): MPI_Wait(request=0x7fffffb658cc,  
                        status0x7fffffff9dd0) failed  
MPI_Wait(76) : Invalid MPI_Request
```

- For MPI/Portals out-of-resources errors, follow advice

```
[193] 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).
```

MPI-IO Improvements

■ MPI-IO collective buffering

⚙️ **MPICH_MPIIO_CB_ALIGN=0**

- ▶ Divides the I/O workload equally among all aggregators
- ▶ Inefficient if multiple aggregators reference the same physical I/O block
- ▶ Default setting in MPT 3.2 and prior versions

⚙️ **MPICH_MPIIO_CB_ALIGN=1**

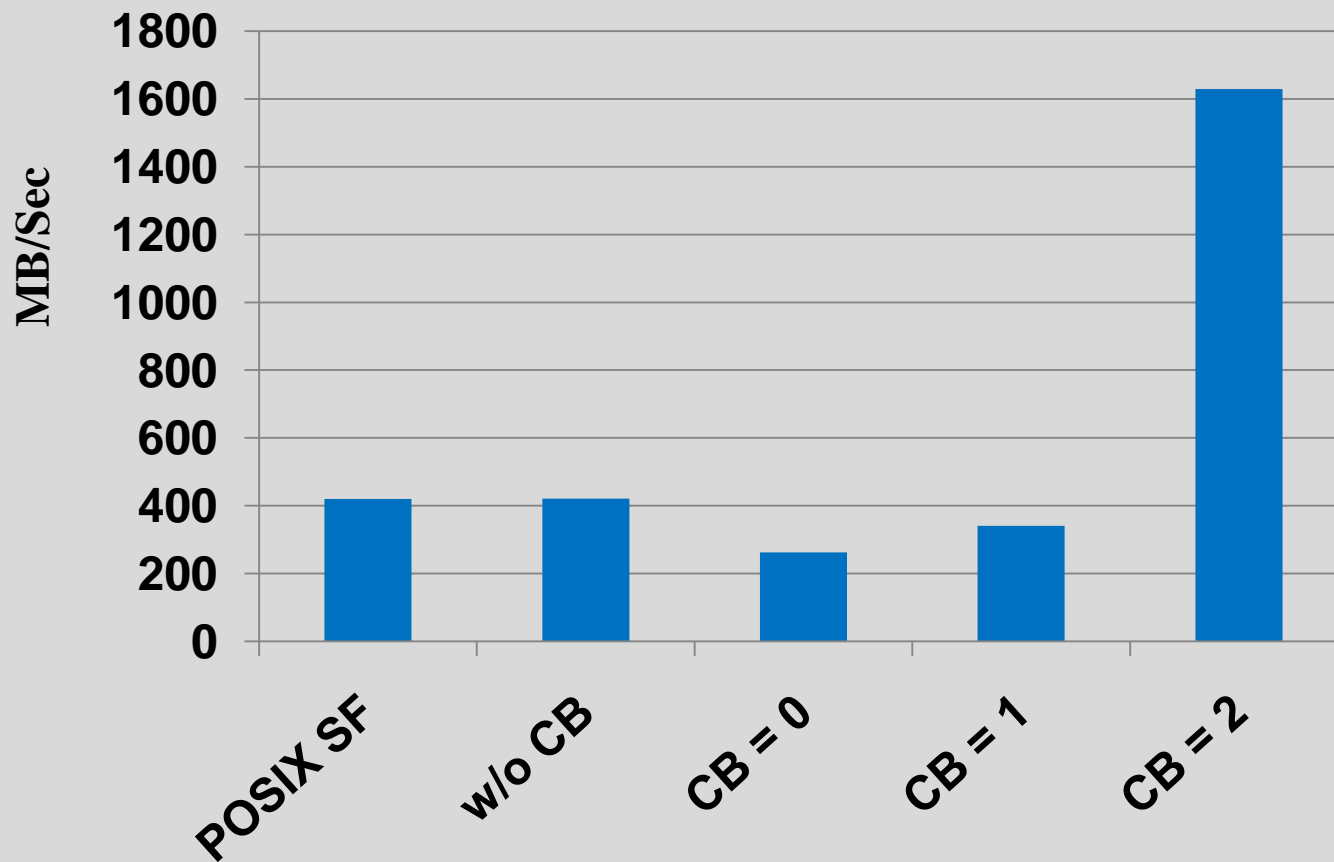
- ▶ Divides the I/O workload up among the aggregators based on physical I/O boundaries and the size of the I/O request
- ▶ Allows only one aggregator access to any stripe on a single I/O call
- ▶ Available in MPT 3.1

⚙️ **MPICH_MPIIO_CB_ALIGN=2**

- ▶ Divides the I/O workload into Lustre stripe-sized groups and assigns them to aggregators
- ▶ Persistent across multiple I/O calls, so each aggregator always accesses the same set of stripes and no other aggregator accesses those stripes
- ▶ Minimizes Lustre file system lock contention
- ▶ Default setting in MPT 3.3

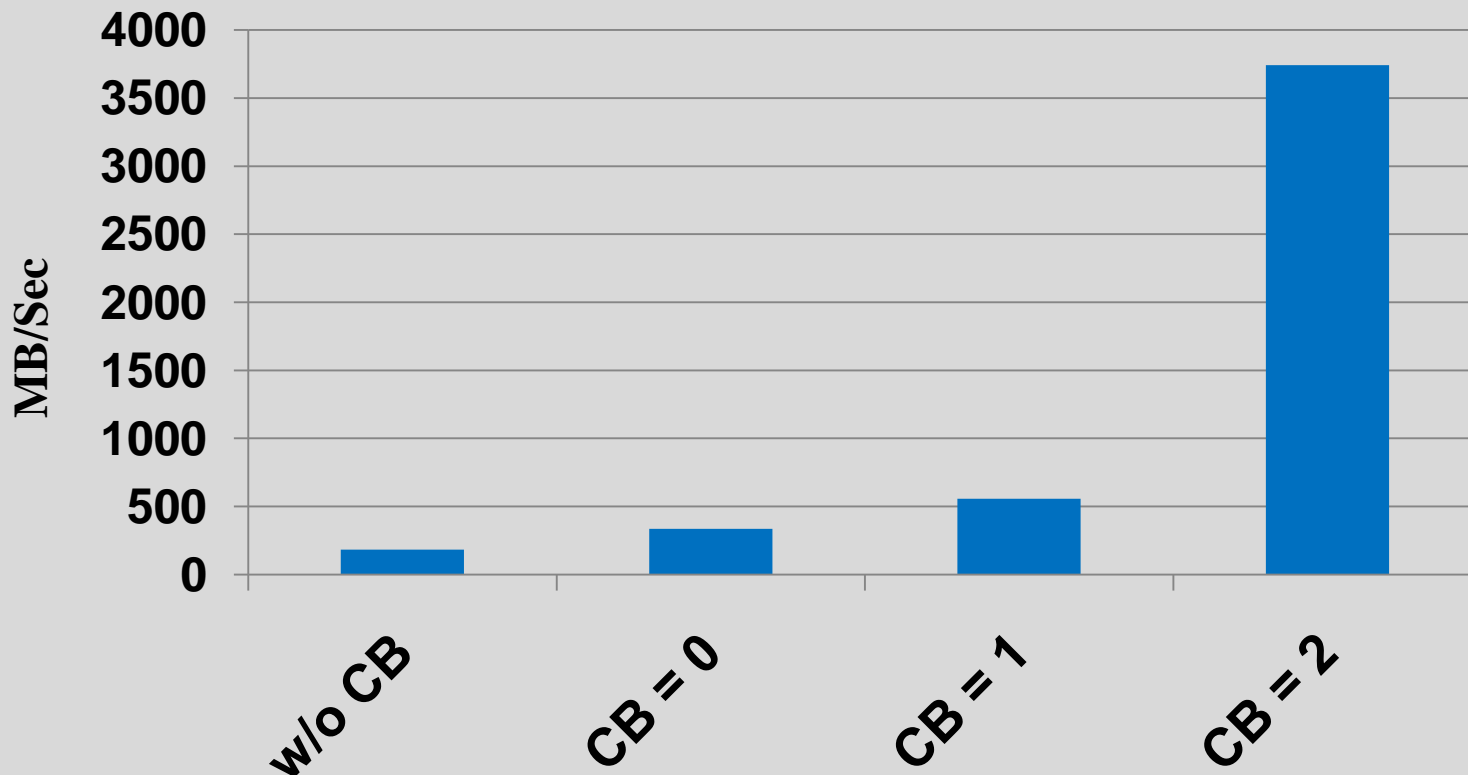
IOR benchmark 1,000,000 bytes

MPI-IO API , non-power-of-2 blocks and transfers, in this case blocks and transfers both of 1M bytes and a strided access pattern. Tested on an XT5 with 32 PEs, 8 cores/node, 16 stripes, 16 aggregators, 3220 segments, 96 GB file



HYCOM MPI-2 I/O

On 5107 PEs, and by application design, a subset of the PEs(88), do the writes. With collective buffering, this is further reduced to 22 aggregators (cb_nodes) writing to 22 stripes. Tested on an XT5 with 5107 PEs, 8 cores/node



Questions . . .