MPI for Cray XE/XK7 Systems

Heidi Poxon
Cray Inc.

Agenda

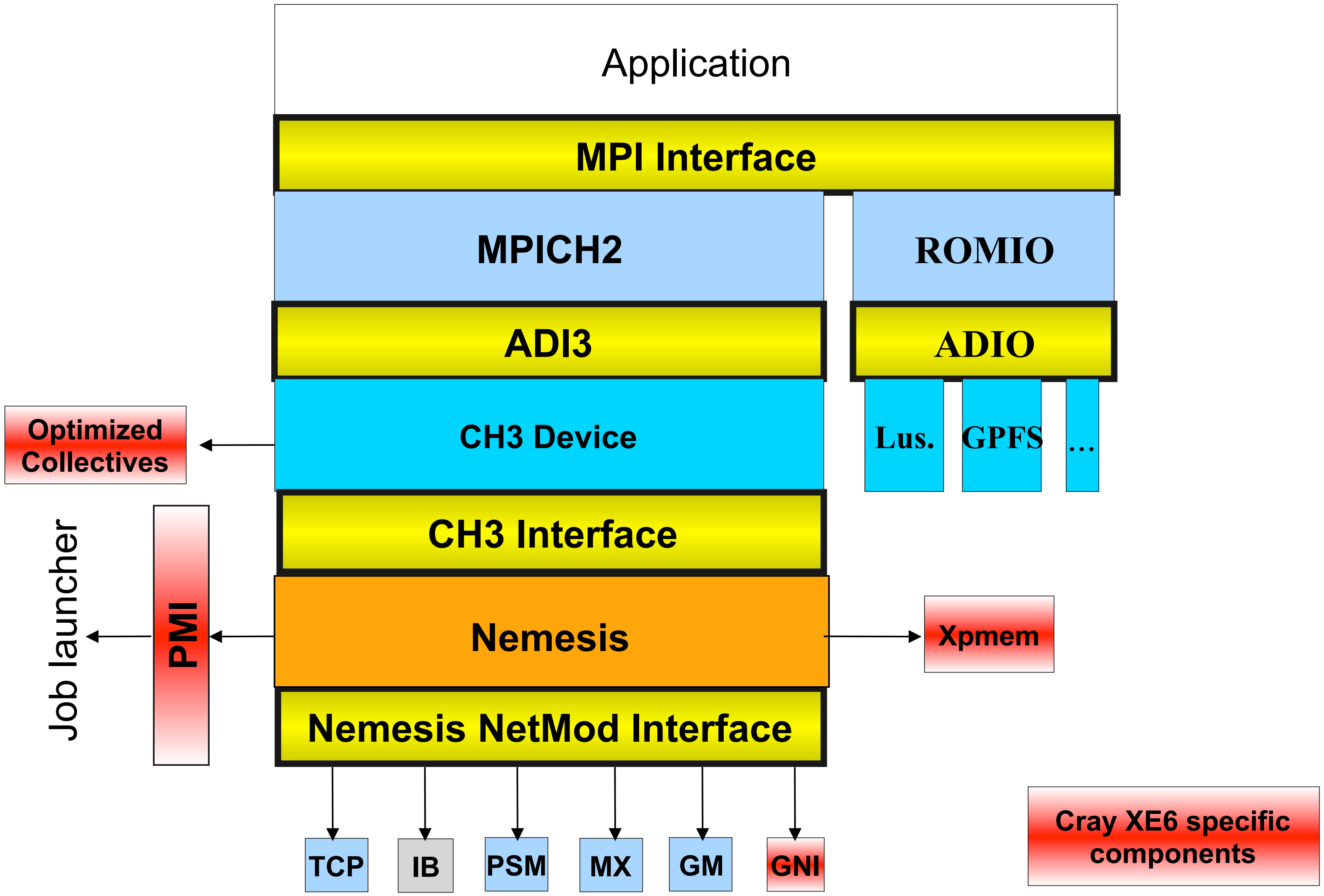
- MPI Overview
- Gemini-specific features used by MPI
- Recent MPI enhancements / differences between ANL MPICH and Cray MPI
- MPI rank re-ordering
- What's coming next

Cray MPI Overview

- **MPT 5.5.5 released October 2012 (default on Titan)**
- **MPT 5.6.0 released November 2012**
- **MPT 5.6.2 coming in February 2013**
  - ORNL received pre-release of cray-mpich2/5.6.2.1
- **ANL MPICH2 version supported: 1.5b1***
- **MPI accessed via cray-mpich2 module (used to be xt-mpich2)**
- **Full MPI2 support (except process spawning) based on ANL MPICH2**
  - Cray uses the MPICH2 Nemesis layer for Gemini
  - Cray provides tuned collectives
  - Cray provides tuned ROMIO for MPI-IO
- ***See intro_mpi man page for details on environment variables, etc.***

*** As of MPT 5.6.0***

MPICH2/Cray layout



Gemini Features Used by MPI

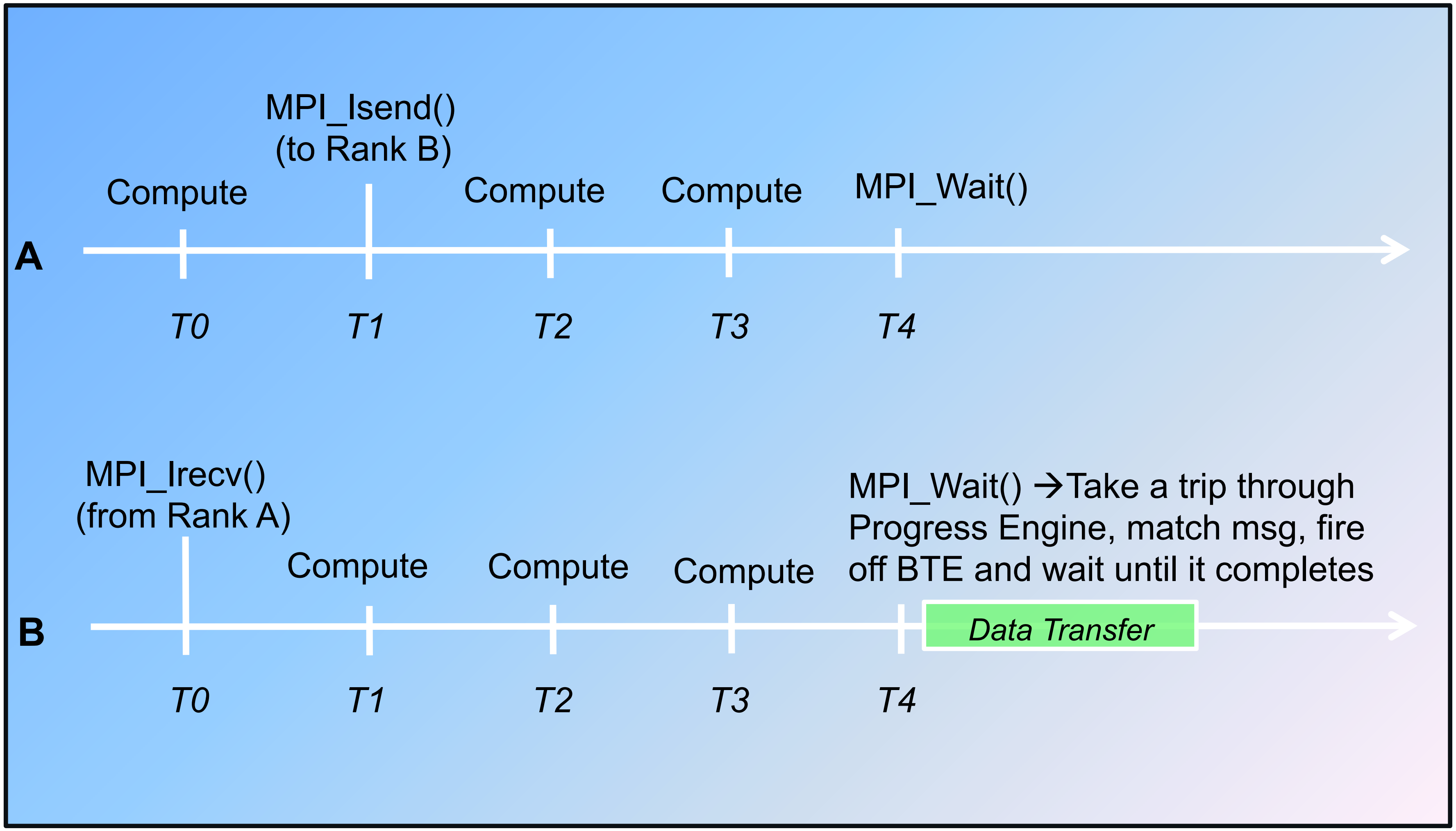
- **FMA (Fast Memory Access)**
  - Used for small messages
  - Called directly from user mode
  - Very low overhead ➔ good latency
- **DMA offload engine (BTE or Block Transfer Engine)**
  - Used for larger messages
  - All ranks on node share BTE resources (4 virtual channels / node)
  - Processed via the OS (no direct user-mode access)
  - Higher overhead to initiate transfer
  - Once initiated, BTE transfers proceed without processor intervention
    - Best means to overlap communication with computation
- **AMOs (Atomic Memory Operations)**
  - Provide a fast synchronization method for collectives

Recent Cray MPI Enhancements

- **Asynchronous Progress Engine**
  - Used to improve communication/computation overlap
  - Each MPI rank starts a "helper thread" during MPI_Init
  - Helper threads progress the MPI state engine while application is computing
  - Only inter-node messages that use Rendezvous Path are progressed (relies on BTE for data motion)
  - Both Send-side and Receive-side are progressed
  - Only effective if used with core specialization to reserve a core/node for the helper threads
  - Must set the following to enable Asynchronous Progress Threads:
    - **export MPICH_NEMESIS_ASYNC_PROGRESS=1**
    - **export MPICH_MAX_THREAD_SAFETY=multiple**
    - Run the application with corespec: **aprun -n XX -r 1 ./a.out**
  - 10% or more performance improvements with some apps

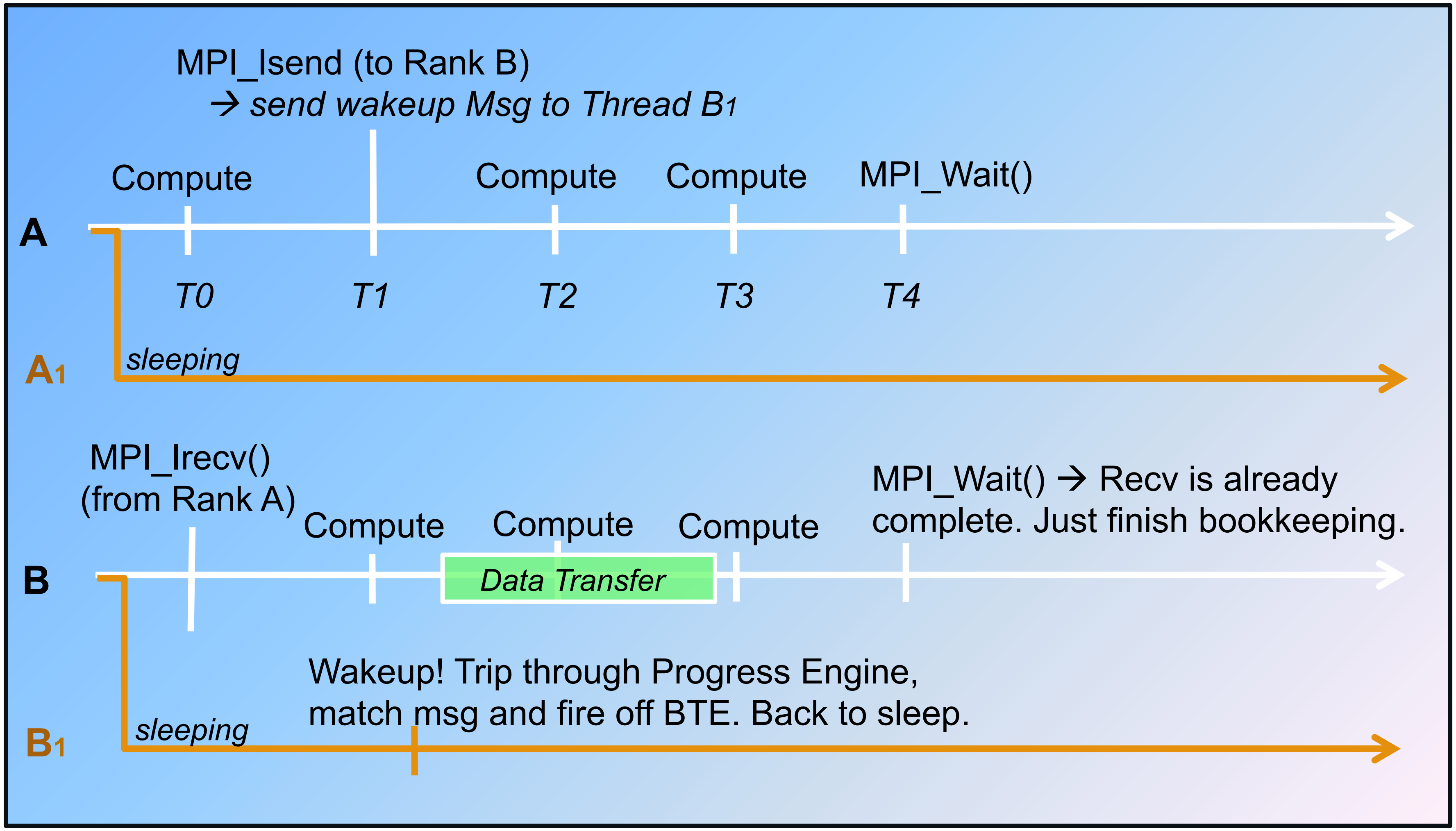
Async Progress Engine Example

2P Example without using Async Progress Threads



Async Progress Engine Example

2P Example using Async Progress Threads


MPI_Isend (to Rank B)
→ send wakeup Msg to Thread B1
Compute
Compute
Compute
MPI_Wait()
A
T0
T1
T2
T3
T4
A1
sleeping
MPI_Irecv()
(from Rank A)
Compute
Compute
Compute
MPI_Wait() → Recv is already complete. Just finish bookkeeping.
B
Data Transfer
Wakeup! Trip through Progress Engine, match msg and fire off BTE. Back to sleep.
B1
sleeping

Recent Cray MPI Enhancements (Cont'd)

Examples of recent collective enhancements:

- **MPI_Gatherv**
  - Replaced poorly-scaling ANL all-to-one algorithm with tree-based algorithm
    - Used if average data size is <=16k bytes
    - MPICH_GATHERV_SHORT_MSG can be used to change cutoff
    - 500X faster than default algorithm at 12,000 ranks with 8 byte messages
- **MPI_Allgather / MPI_Allgatherv**
  - Optimized to access data efficiently for medium to large messages (4k – 500k bytes)
  - 15% to 10X performance improvement over default MPICH2
- **MPI_Barrier**
  - Uses DMAPP GHAL collective enhancements
    - To enable set: **export MPICH_USE_DMAPP_COLL=1**
    - Requires DMAPP (libdmapp) be linked into the executable
    - Internally dmapp_init is called (may require hugepages, more memory)
    - Nearly 2x faster than default MPICH2 Barrier
- **Improved MPI_Scatterv algorithm for small messages***
  - Significant improvement for small messages on very high core counts
  - See MPICH_SCATTERV_SHORT_MSG for more info
  - Over 15X performance improvement in some cases

MPI Collectives Optimized for XE/XK

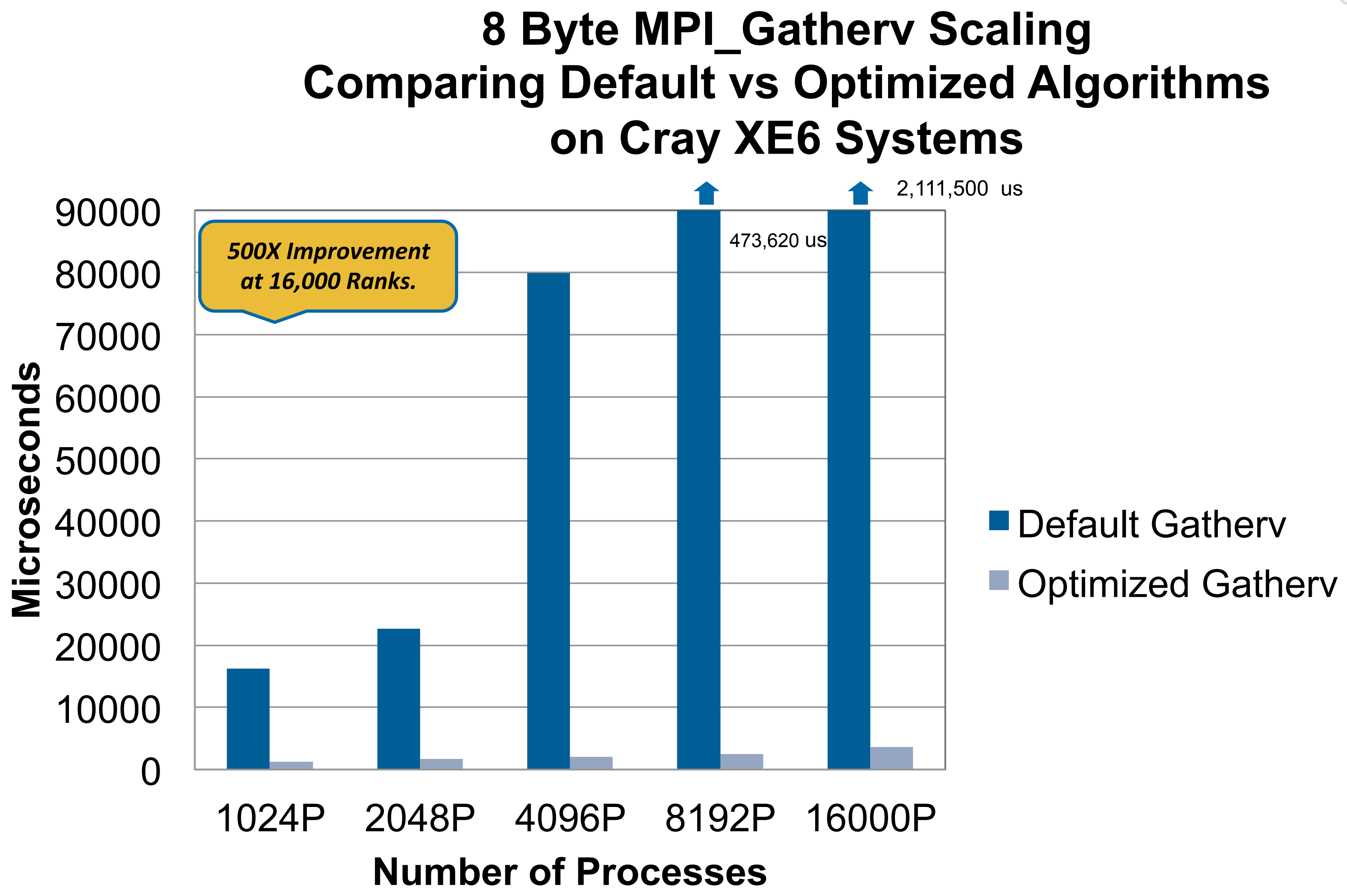
Optimizations **on** by default unless specified for:

- MPI_Alltoall
- MPI_Alltoallv
- MPI_Bcast
- MPI_Gather
- MPI_Gatherv
- MPI_Allgather
- MPI_Allgatherv
- MPI_Scatterv

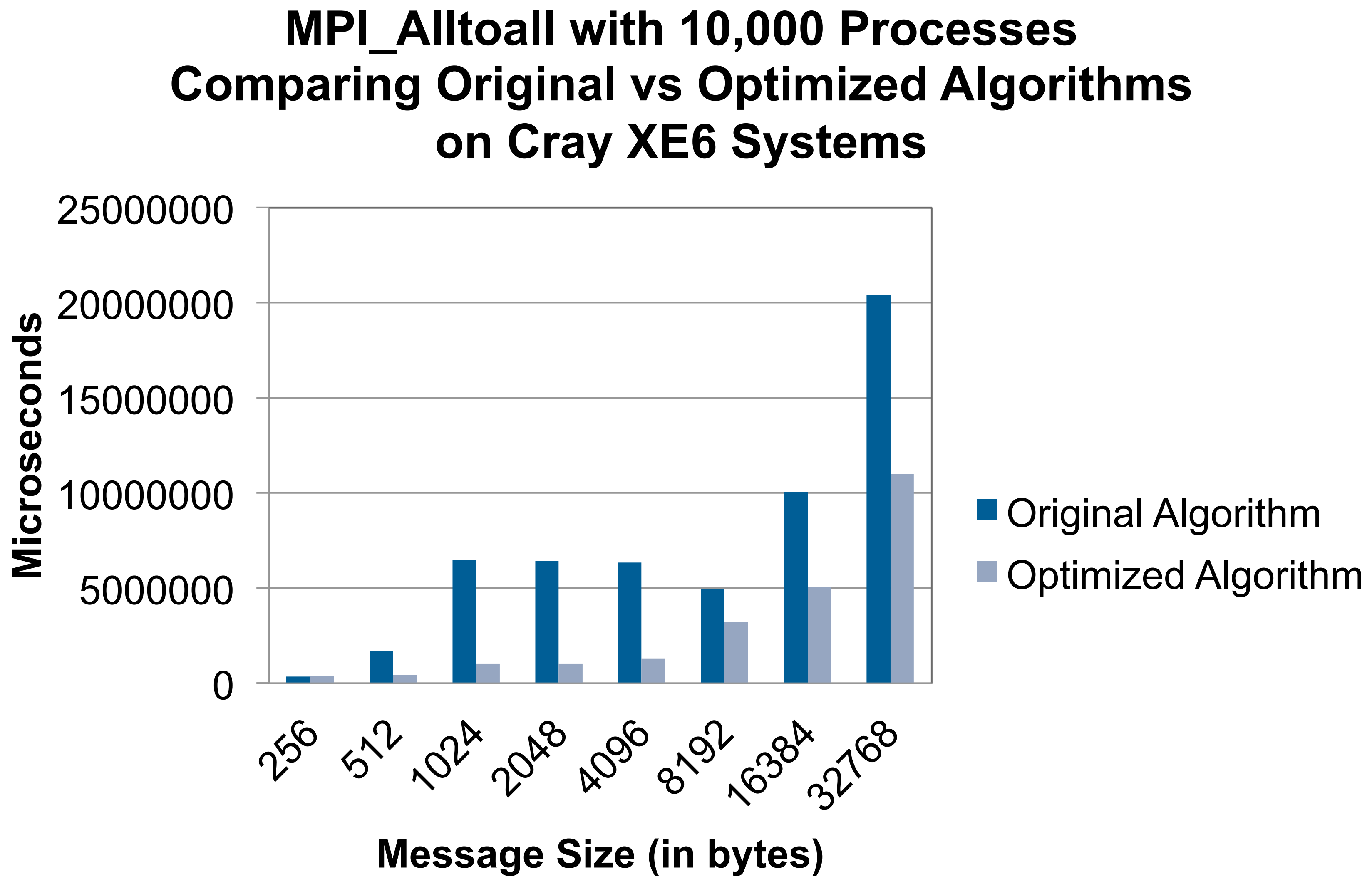
Optimizations **off** by default unless specified for

- **MPI_Allreduce** and **MPI_Barrier**
  - These two use DMAPP GHAL enhancements. *Not enabled by default*.
  - export **MPICH_USE_DMAPP_COLL=1**

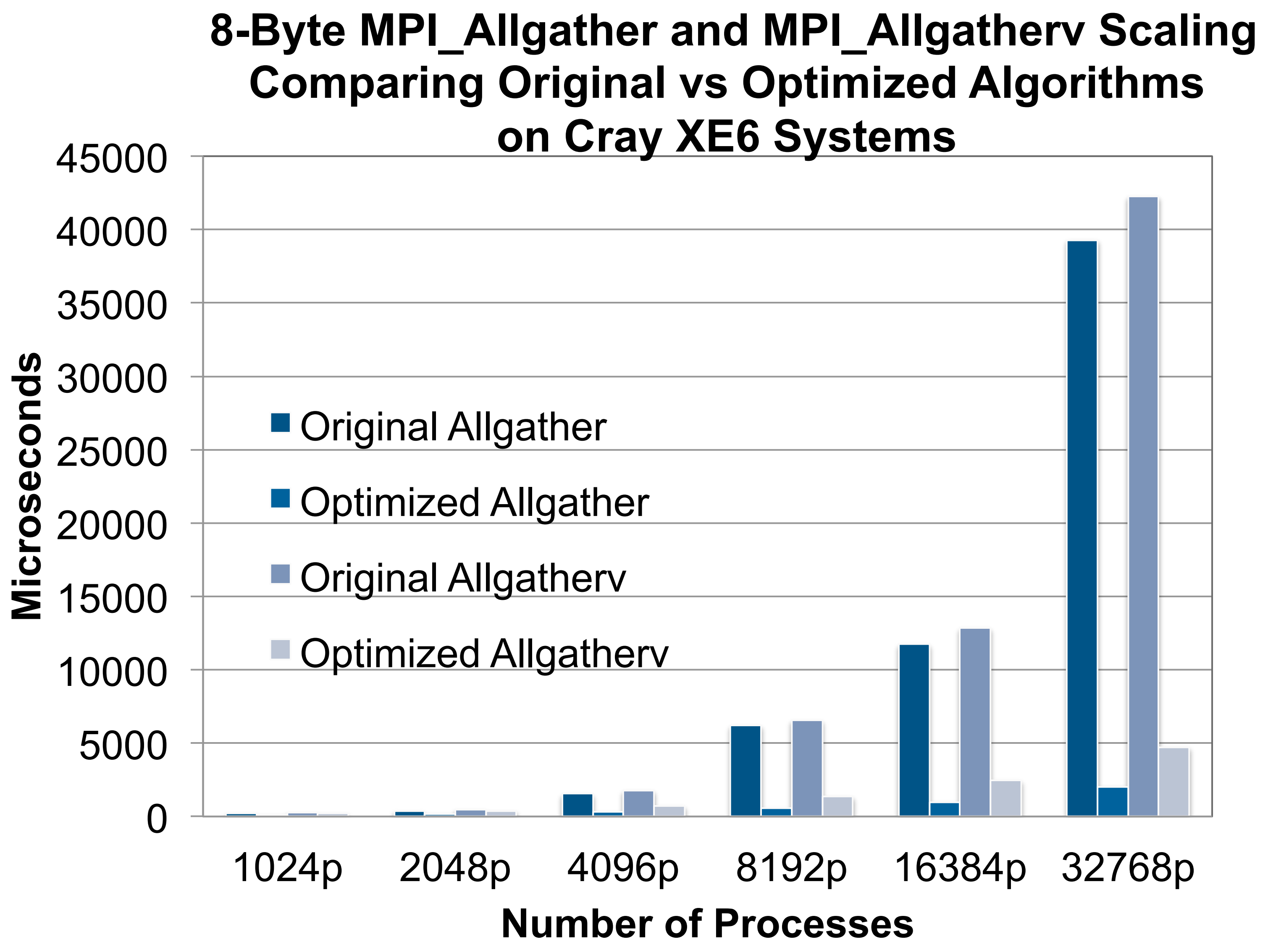
MPI_Gatherv Performance



Improved MPI_Alltoall



MPI_Allgather Improvements



Recent Cray MPI Enhancements (Cont'd)

- **Minimize MPI memory footprint**
  - Optional mode to allow fully connected pure-MPI jobs to run across large number of cores
  - Memory usage slightly more than that seen with only 1 MPI rank per node
  - See MPICH_GNI_VC_MSG_PROTOCOL env variable
  - May reduce performance significantly but will allow some jobs to run that could not otherwise
- **Static vs dynamic connection establishment**
  - Optimizations for performance improvements to both modes
  - Static mode most useful for codes that use MPI_Alltoall
  - See MPICH_GNI_DYNAMIC_CONN env variable

Recent Cray MPI Enhancements (Cont'd)

- **MPI-3 non-blocking collectives available as MPIX_ functions***
  - Reasonable overlap seen for messages more than 16K bytes, 8 or less ranks per node and at higher scale
  - Recommend to use core-spec (aprun –r option) and setting MPICH_NEMESIS_ASYNC_PROGRESS=1 and MPICH_MAX_THREAD_SAFETY=multiple
- **MPI I/O file access pattern statistics***
  - When setting MPICH_MPIIO_STATS=1, a summary of file write and read access patterns are written by rank 0 to stderr
  - Information is on a per-file basis and written when the file is closed
  - The "Optimizing MPI I/O" white paper describes how to interpret the data and makes suggestions on how to improve your application.
    - Available on docs.cray.com under Knowledge Base
- **Improved overall scaling of MPI to over 700K MPI ranks**
  - Number of internal mailboxes now dependent on the number of ranks in the job. See MPICH_GNI_MBOXES_PER_BLOCK env variable for more info
  - Default value of MPICH_GNI_MAX_VSHORT_MSG_SIZE now set to 100 bytes for programs using more than 256K MPI ranks. This is needed to reduce the size of the pinned mailbox memory for static allocations.

MPI Rank Order

Is your nearest neighbor really your nearest neighbor?

And do you want them to be your nearest neighbor?

MPI Rank Placement

- **Change default rank ordering with:**
  - MPICH_RANK_REORDER_METHOD
- **Settings:**
  - 0: Round-robin placement – Sequential ranks are placed on the next node in the list. Placement starts over with the first node upon reaching the end of the list.
  - 1: SMP-style placement – Sequential ranks fill up each node before moving to the next. - DEFAULT
  - 2: Folded rank placement – Similar to round-robin placement except that each pass over the node list is in the opposite direction of the previous pass.
  - 3: Custom ordering - The ordering is specified in a file named MPICH_RANK_ORDER.

When Is Rank Re-ordering Useful?

- Maximize on-node communication between MPI ranks
- Grid detection and rank re-ordering is helpful for programs with significant point-to-point communication
- Relieve on-node shared resource contention by pairing threads or processes that perform different work (for example computation with off-node communication) on the same node

Automatic Communication Grid Detection

- **Cray performance tools produce a custom rank order if it's beneficial based on grid size, grid order and cost metric**
- **Heuristics available for:**
  - MPI sent message statistics
  - User time (time spent in user functions) – can be used for PGAS codes
  - Hybrid of sent message and user time)
- **Summarized findings in report**
- **Available with sampling or tracing**
- **Describe how to re-run with custom rank order**

MPI Rank Order Observations

```
Table 1:  Profile by Function Group and Function

 Time%  |      Time   |      Imb. |    Imb. |    Calls  |Group
        |             |      Time |   Time% |           | Function
        |             |           |         |           |  PE=HIDE

 100.0% | 463.147240 |        -- |      -- | 21621.0 |Total
|-----------------------------------------------------------------------
|  52.0% | 240.974379 |        -- |      -- | 21523.0 |MPI
||----------------------------------------------------------------------
||  47.7% | 221.142266 | 36.214468 |   14.1% | 10740.0 |mpi_recv
||   4.3% |  19.829001 | 25.849906 |   56.7% | 10740.0 |MPI_SEND
||======================================================================
|  43.3% | 200.474690 |        -- |      -- |    32.0 |USER
||----------------------------------------------------------------------
||  41.0% | 189.897060 | 58.716197 |   23.6% |    12.0 |sweep_
||   1.6% |   7.579876 |  1.899097 |   20.1% |    12.0 |source_
||======================================================================
|   4.7% |  21.698147 |        -- |      -- |    39.0 |MPI_SYNC
||----------------------------------------------------------------------
|   4.3% |  20.091165 | 20.005424 |   99.6% |    32.0 | mpi_allreduce_(sync)
||======================================================================
|   0.0% |   0.000024 |        -- |      -- |    27.0 |SYSCALL
|=======================================================================
```

MPI Rank Order Observations (2)

```
MPI Grid Detection:

    There appears to be point-to-point MPI communication in a 96 X 8
    grid pattern. The 52% of the total execution time spent in MPI
    functions might be reduced with a rank order that maximizes
    communication between ranks on the same node. The effect of several
    rank orders is estimated below.

    A file named MPICH_RANK_ORDER.Grid was generated along with this
    report and contains usage instructions and the Custom rank order
    from the following table.

         Rank    On-Node     On-Node   MPICH_RANK_REORDER_METHOD
        Order    Bytes/PE   Bytes/PE%
                             of Total
                             Bytes/PE

        Custom   2.385e+09     95.55%  3
           SMP   1.880e+09     75.30%  1
          Fold   1.373e+06      0.06%  2
    RoundRobin   0.000e+00      0.00%  0
```

MPICH_RANK_ORDER File

```
# The 'Custom' rank order in this file targets nodes with multi-core
# processors, based on Sent Msg Total Bytes collected for:
#
# Program:      /lus/nid00030/heidi/sweep3d/mod/sweep3d.mpi
# Ap2 File:     sweep3d.mpi+pat+27054-89t.ap2
# Number PEs:   48
# Max PEs/Node: 4
#
# To use this file, make a copy named MPICH_RANK_ORDER, and set the
# environment variable MPICH_RANK_REORDER_METHOD to 3 prior to
# executing the program.
#
# The following table lists rank order alternatives and the grid_order
# command-line options that can be used to generate a new order.
…
```

Auto-Generated MPI Rank Order File

```
# The
'USER_Time_hybrid'
rank order in this
file targets nodes
with multi-core
# processors, based on
Sent Msg Total Bytes
collected for:
#
# Program:      /lus/
nid00023/malice/
craypat/WORKSHOP/bh2o-
demo/Rank/sweep3d/src/
sweep3d
# Ap2 File:
sweep3d.gmpi-u.ap2
# Number PEs:   768
# Max PEs/Node:  16
#
# To use this file,
make a copy named
MPICH_RANK_ORDER, and
set the
# environment variable
MPICH_RANK_REORDER_MET
HOD to 3 prior to
# executing the
program.
#
0,532,64,564,32,572,96
,540,8,596,72,524,40,6
04,24,588
104,556,16,628,80,636,
56,620,48,516,112,580,
88,548,120,612

1,403,65,435,33,411,97
,443,9,467,25,499,105,
507,41,475
73,395,81,427,57,459,1
7,419,113,491,49,387,8
9,451,121,483
6,436,102,468,70,404,3
8,412,14,444,46,476,11
0,508,78,500
86,396,30,428,62,460,5
4,492,118,420,22,452,9
4,388,126,484
129,563,193,531,161,57
1,225,539,241,595,233,
523,249,603,185,555
153,587,169,627,137,63
5,201,619,177,515,145,
579,209,547,217,611
7,405,71,469,39,437,10
3,413,47,445,15,509,79
,477,31,501
111,397,63,461,55,429,
87,421,23,493,119,389,
95,453,127,485
134,402,198,434,166,41
0,230,442,238,466,174,
506,158,394,246,474
190,498,254,426,142,45
8,150,386,182,418,206,
490,214,450,222,482
128,533,192,541,160,56
5,232,525,224,573,240,
597,184,557,248,605
168,589,200,517,152,62
9,136,549,176,637,144,
621,208,581,216,613

5,439,37,407,69,447,10
1,415,13,471,45,503,29
,479,77,511
53,399,85,431,21,463,6
1,391,109,423,93,455,1
17,495,125,487
2,530,34,562,66,538,98
,522,10,570,42,554,26,
594,50,602
18,514,74,586,58,626,8
2,546,106,634,90,578,1
14,618,122,610
135,315,167,339,199,34
7,259,307,231,371,239,
379,191,331,247,299
175,363,159,323,143,35
5,255,291,207,275,183,
283,151,267,215,223
133,406,197,438,165,47
0,229,414,245,446,141,
478,237,502,253,398
157,510,189,462,173,43
0,205,390,149,422,213,
454,181,494,221,486
130,316,260,340,194,37
2,162,348,226,308,234,
380,242,332,250,300
202,364,186,324,154,35
6,138,292,170,276,178,
284,210,218,268,146
4,535,36,543,68,567,10
0,527,12,599,44,575,28
,559,76,607
52,591,20,631,60,639,8
4,519,108,623,92,551,1
16,583,124,615

3,440,35,432,67,400,99
,408,11,464,43,496,27,
472,51,504
19,392,75,424,59,456,8
3,384,107,416,91,488,1
15,448,123,480
132,401,196,441,164,40
9,228,433,236,465,204,
473,244,393,188,497
252,505,140,425,212,45
7,156,385,172,417,180,
449,148,489,220,481
131,534,195,542,163,56
6,227,526,235,574,203,
598,243,558,187,606
251,590,211,630,179,63
8,139,622,155,550,171,
518,219,582,147,614
761,660,737,652,705,66
8,745,692,673,700,641,
684,713,644,753,724
729,732,681,756,721,71
6,764,676,697,748,689,
657,740,665,649,708
760,528,736,536,704,56
0,744,520,672,568,712,
592,752,552,640,600
728,584,680,624,720,51
2,696,632,688,616,664,
544,608,656,648,576
762,659,738,651,706,66
7,746,643,714,691,674,
699,754,683,730,723
722,731,763,658,642,75
5,739,675,707,650,682,
715,698,666,690,747

257,345,265,313,281,30
5,273,337,609,369,577,
377,617,329,513,529
545,297,633,361,625,32
1,585,537,601,289,553,
353,593,521,569,561
256,373,261,341,264,34
9,280,317,272,381,269,
309,285,333,277,365
352,301,320,325,288,35
7,328,304,360,312,376,
293,296,368,336,344
258,338,266,346,282,31
4,274,370,766,306,710,
378,742,330,678,362
646,298,750,322,718,35
4,758,290,734,662,686,
670,726,702,694,654
262,375,263,343,270,31
1,271,351,286,319,278,
342,287,350,279,374
294,318,358,383,359,31
0,295,382,326,303,327,
367,366,335,302,334
765,661,709,663,741,65
3,711,669,767,655,743,
671,749,695,679,703
677,727,751,693,647,70
1,717,687,757,685,733,
725,719,735,645,759
```

grid_order Utility

- Can use grid_order utility without first running the application with the Cray performance tools if you know a program’s data movement pattern
- Originally designed for MPI programs, but since reordering is done by PMI, it can be used by other programming models (since PMI is used by MPI, SHMEM and PGAS programming models)
- Utility available if perftools modulefile is loaded
- See grid_order(1) man page or run grid_order with no arguments to see usage information

Reorder Example for Bisection Bandwidth

- Assume 32 ranks
- Decide on row or column ordering:
- $ grid_order –R –g 2,16

```
0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15
16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31
```

- $ grid_order –C –g 2,16

```
0,2,4,6,8,10,12,14,16,18,20,22,24,26,28,30
1,3,5,7,9,11,13,15,17,19,21,23,25,27,29,31
```

- Since rank 0 talks to rank 16, and not with rank 1, we choose Row ordering

Reorder Example for Bisection Bandwidth (2)

- Specify cell (or chunk) to make sure rank pairs live on same node (but don't care how many pairs live on a node)
- $ grid_order –R –g 2,16 –c 2,1

```
0,16
1,17
2,18
3,19
4,20
5,21
6,22
7,23
8,24
9,25
10,26
11,27
12,28
13,29
14,30
15,31
```

Fills a Magny-Cours node

Using New Rank Order

- Save grid_order output to file called MPICH_RANK_ORDER
- Export MPICH_RANK_REORDER_METHOD=3
- Run non-instrumented binary with and without new rank order to check overall wallclock time for improvements

Example Performance Results for upcbw

- **Default thread ordering**
  - Application 8538980 resources: utime ~126s, stime ~108s

- **Maximized on-node data movement with reordering**
  - Application 8538982 resources: utime ~38s, stime ~106s

Case Study

- **AWP-ODC code from NCAR procurement**
  - Earthquake code – x, y, z structured grid
- **MPI uses different mechanisms for on-node and off-node communication**
  - Shared memory on node – fast
  - uGNI between nodes – not as fast
- **AWP-ODC grid => 3-D grid of blocks**
  - Each block mapped to a processor
  - Map blocks to node to minimize off-node communication
- **Use MPI rank re-ordering to map blocks to nodes**

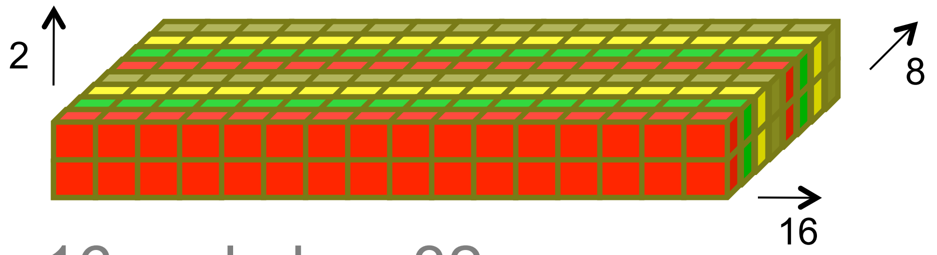
AWP-ODC and grid_order

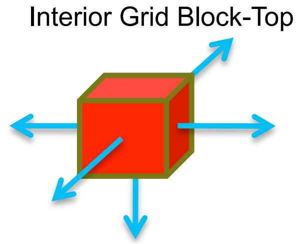
- **If MPICH_RANK_REORDER_METHOD=3**
  then rank order => MPICH_RANK_ORDER file
- **Use grid_order to generate MPICH_RANK_ORDER**
  - Part of perftools
  - "module load perftools" to access command/man-page
  - grid_order –C –g x,y,z –c nx, ny, nz
    - -C: row major ordering
    - -g: x, y, z grid size
      - x*y*z = number of MPI processes
    - -c: nx, ny, nz of the grids on node
      - nx*ny*nz = number of MPI processes on a node
  - MPI re-order file written to stdout

AWP-ODC example – Part 1

- **NCAR provided three test cases:**
  - 256 processors: 16x2x8 grid
  - 512 processors: 16x4x8 grid
  - 1024 processors: 16x4x16 grid
- **For 256 processors: 16x2x8 grid**

Graphics by Kevin McMahon





  - IL-16 node has 32 cores
    - Possible grid block groups (nx, ny, nz) for a node:
    - 16x2x1: 64 neighbors off-node
    - 2x2x8: 32 neighbors off-node
    - 4x2x4: 24 neighbors off -node

AWP-ODC example – Part 2

- **For 256 processors test case**
  - Using 2x2x8 blocks/node was fastest
    - Default: 0.097 sec/compute iter
    - 2x2x8 blocks/node: 0.085 sec/compute iter
  - 12% faster than the default results!

- **Final additions to the 256pe PBS batch script:**

```
. ${MODULESHOME}/init/sh
module load perftools
export MPICH_RANK_REORDER_METHOD=3
/bin/rm –rf MPICH_RANK_ORDER

grid_order –C –g 16,2,8 –c 2,2,8 > MPICH_RANK_ORDER
```

What’s Coming Next?

- GPU-to-GPU support
- Merge to MPICH 3.0 release from ANL
- Release and optimize MPI-3 features
- Improvements to small message MPI_Alltoall at scale
- Improvements to MPI I/O
- MPI Stats / Bottlenecks Display

GPU-to-GPU Optimization Feature

- Coming in February 2013
- Set MPICH_RDMA_ENABLED_CUDA=1
- Pass GPU pointer directly to MPI point-to-point or collectives

Example without GPU-to-GPU...

```
if (rank == 0) {
    // Copy from device to host, then send.
    cudaMemcpy(host_buf, device_buf, …);
    MPI_Send(host_buf, …);
} else if (rank == 1) {
    // Receive, then copy from host to device.
    MPI_Recv(host_buf,...);
    cudaMemcpy(device_buf, host_buf,...);
}
```

Example with GPU-to-GPU...

```
if (rank == 0) {
    // Send device buffer.
    MPI_Send(device_buf, …);
} else if (rank == 1) {
    // Receive device buffer.
    MPI_Recv(device_buf,...);
}
```

GPU-to-GPU Optimization Specifics

- Under the hood (i.e., in the GNI netmod), GPU-to-GPU messages are pipelined to improve performance (only applies to long message transfer aka rendezvous messages)
- The goal is to overlap communication between the GPU and the host, and the host and the NIC
- Ideally, this would hide one of the two memcpy's
- We see up to a 50% performance gain.

GPU-to-GPU optimization (Cont’d)

- On the send side (similar for recv. side)...
- Data is prefetched from the GPU using cudaMemcpyAsync.
- Data that has already been transferred to the host is sent over the network (this is off-loaded to the BTE engine).
- This allows for overlap between communication and computation.

Example GPU-to-GPU overlap

Since asynchronous cudaMemcpy's are used internally, it makes sense to do something like this...

```
if (rank == 0) {
    MPI_Isend(device_buf, …, &sreq);
    while (work_to_do) [do some work]
    MPI_Wait(&sreq, MPI_STATUS_IGNORE);
} else if (rank == 1)
    MPI_Irecv(device_buf,..., &rreq);
    while (nothing_better_to_do) [do some work]
    MPI_Wait(&rreq, MPI_STATUS_IGNORE);
}
```

Summary

- Cray MPI optimizations based on message transfer size and job size
- Cray works very hard to establish good defaults
- Should be able to get very good overlap of large pt2pt messages (use async progress engine with core specialization)
- Understand where your performance bottleneck is
  - If MPI_Alltoall for example,
    - Look at Alltoall-specific environment variables (man intro_mpi(3))
    - Try the non-blocking collectives
  - If communication load imbalance detected
    - Try custom rank reorder

A Day in the Life of an MPI Inter-Node Message

MPI Inter-Node Messaging

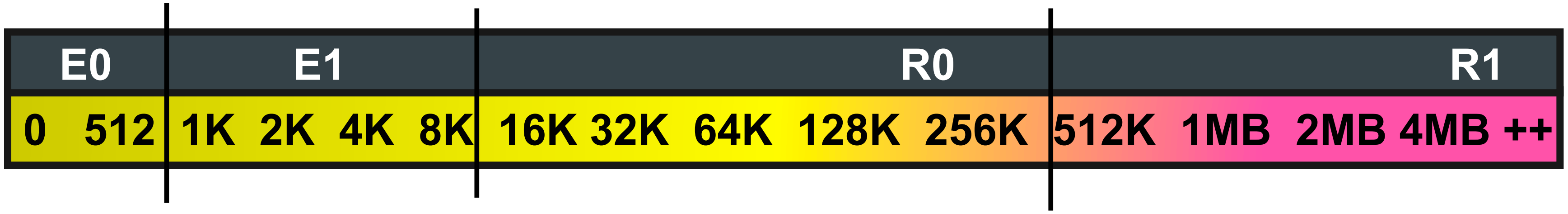
- Four message protocols based on size of message…
- Eager Message Protocol (up to 8K bytes)
  - E0 and E1 Paths
- Rendezvous Message Protocol
  - R0 and R1 Paths
- MPI environment variables that alter those paths

Environment Variables that Affect Protocols

- **MPICH_GNI_MAX_VSHORT_MSG_SIZE**
  - Controls max size for E0 path
  - Default varies with job size: 216-984 bytes
- **MPICH_GNI_MAX_EAGER_MSG_SIZE**
  - Controls max message size for E1 path (default: 8K bytes)
- **MPICH_GNI_NDREG_MAXSIZE**
  - Controls max message size for R0 path (default: 4MB)
- **MPICH_GNI_LMT_PATH=disabled**
  - Can be used to disable entire rendezvous path
  - MPI falls back to using internal buffers for long message transfers, disabling zero-copy RDMA protocols

MPI Inter-node Messaging

- **Four Main Pathways through the MPICH2 GNI NetMod**
  - Two EAGER paths (E0 and E1)
  - Two RENDEZVOUS (aka LMT) paths (R0 and R1)
- **Selected Pathway is Based (generally) on Message Size**



- **MPI env variables affecting the pathway**
  - **MPICH_GNI_MAX_VSHORT_MSG_SIZE**
    - Controls max size for E0 Path (Default varies with job size: 216-8152 bytes)
  - **MPICH_GNI_MAX_EAGER_MSG_SIZE**
    - Controls max message size for E1 Path (Default is 8K bytes)
  - **MPICH_GNI_NDREG_MAXSIZE**
    - Controls max message size for R0 Path (Default is 512K bytes)
  - **MPICH_GNI_LMT_PATH=disabled**
    - Can be used to Disable the entire Rendezvous (LMT) Path

EAGER Message Protocol

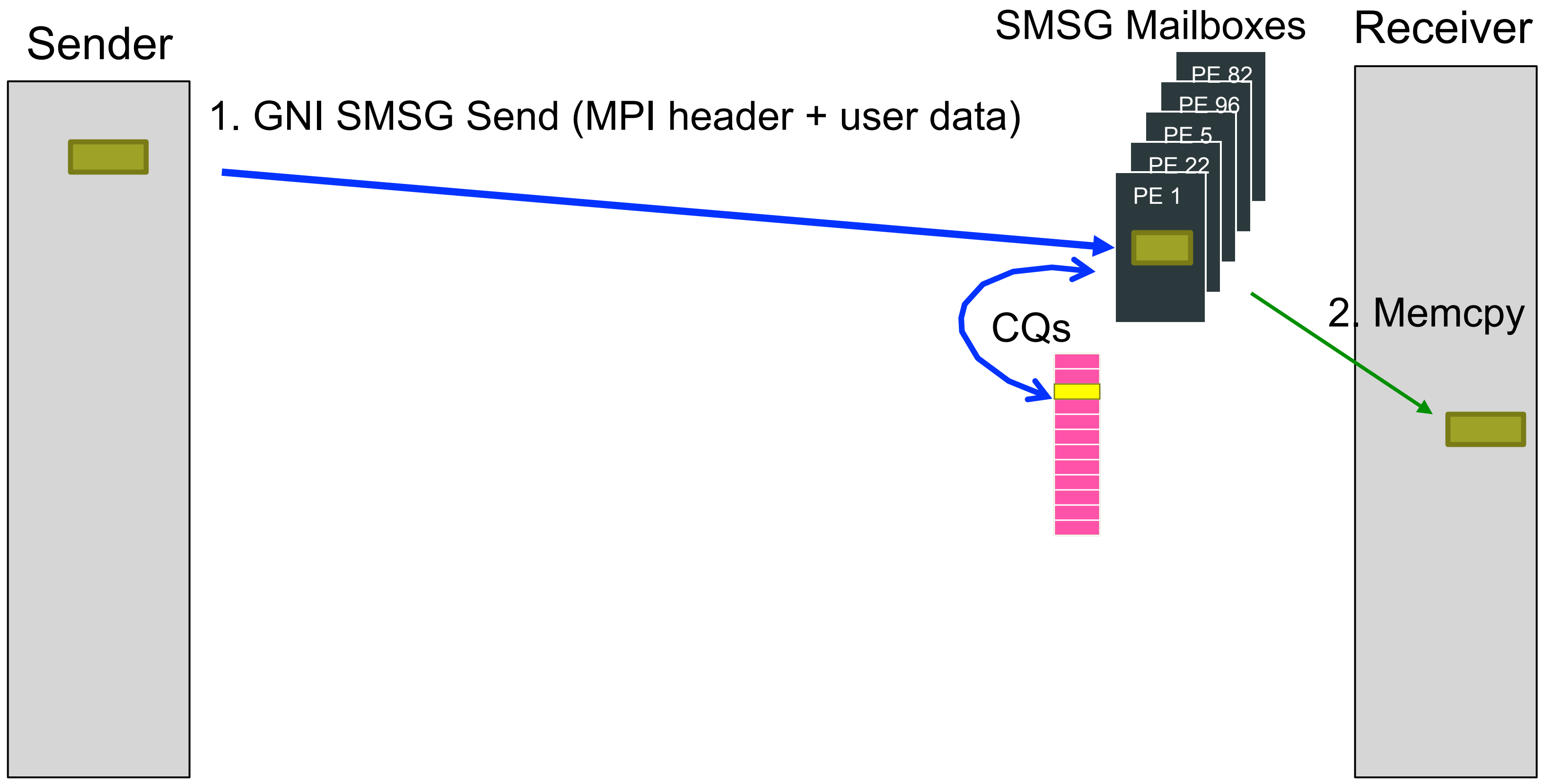
- **Data is transferred when MPI_Send (or variant) encountered**
  - Implies data will be buffered on receiver's node
- **Two EAGER Pathways**
  - **E0** – small messages that fit into GNI SMSG Mailbox
    - Default mailbox size varies with number of ranks in the job

Job Size	Max User Data (in bytes)
1 < ranks <= 512	8152
512 < ranks <= 1024	2008
1024 < ranks < 16384	472
16384 < ranks < 256K	216

    - Use **MPICH_GNI_MAX_VSHORT_MSG_SIZE** to adjust size
  - **E1** – too big for SMSG Mailbox, but small enough to still go EAGER
    - Use **MPICH_GNI_MAX_EAGER_MSG_SIZE** to adjust size
    - Requires extra copies

MPI Inter-Node Message type E0

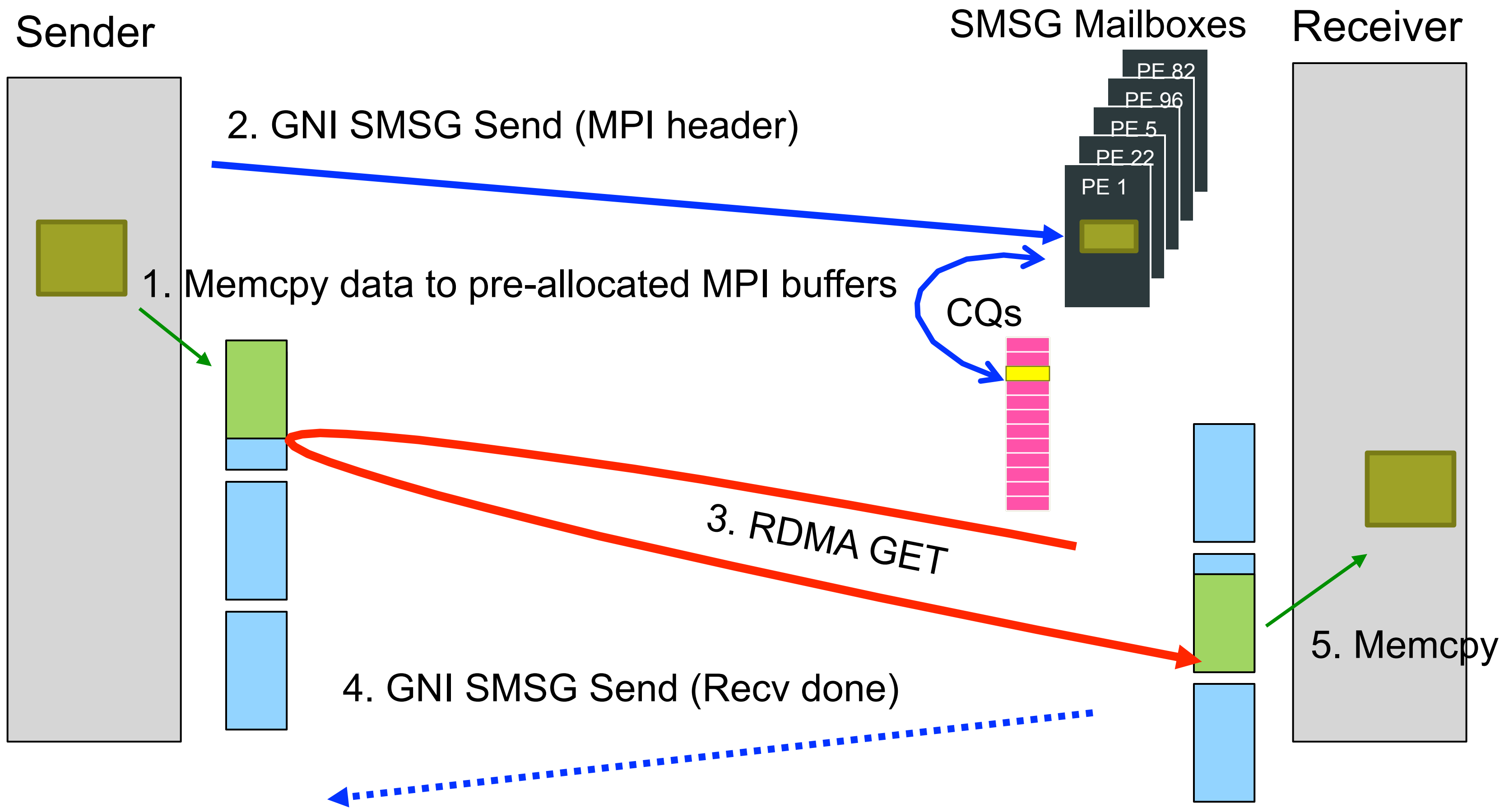
EAGER messages that fit in the GNI SMSG Mailbox



- GNI SMSG Mailbox size changes with the number of ranks in the job
- Mailboxes use large pages by default (even if app isn't using them itself)

MPI Inter-Node Message type E1

EAGER messages that don't fit in the GNI SMSG Mailbox



Sender
SMSG Mailboxes
Receiver
PE 82
PE 96
PE 5
PE 22
PE 1
2. GNI SMSG Send (MPI header)
1. Memcpy data to pre-allocated MPI buffers
CQs
3. RDMA GET
4. GNI SMSG Send (Recv done)
5. Memcpy

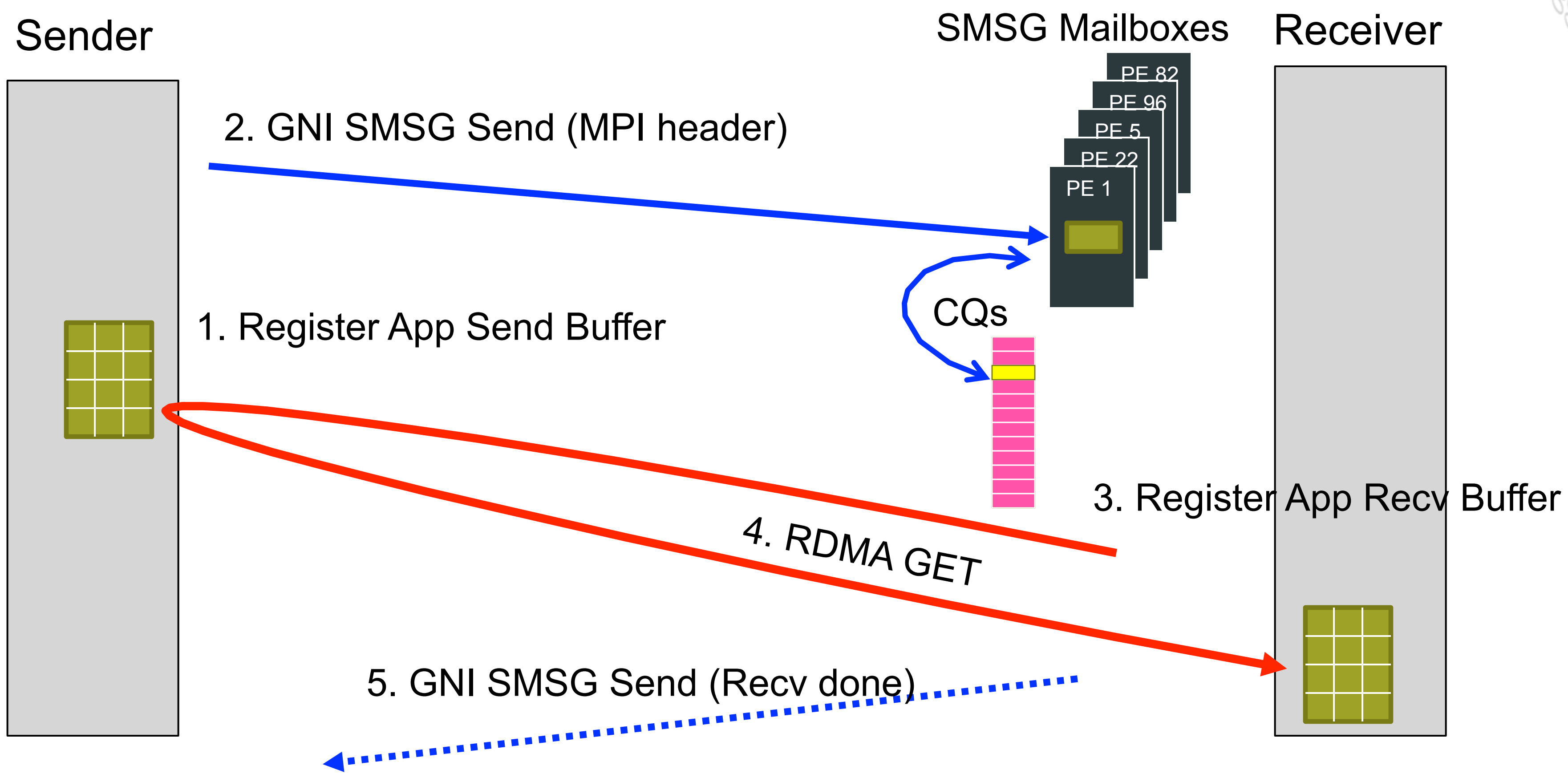
- User data is copied into internal MPI buffers on both send and receive side
- Default MPICH_GNI_NUM_BUFS is 64 (each buffer is 32K)
- Internal MPI buffers use large pages

RENDEZVOUS Message Protocol

- **Data is transferred after receiver has posted matching receive for a previously initiated send**
- **Two RENDEZVOUS Pathways**
  - R0 – **RDMA GET** method
    - By default, used for messages between 8K and 4 MB
    - Use **MPICH_GNI_MAX_EAGER_MSG_SIZE** to adjust starting point
    - Use **MPICH_GNI_NDREG_MAXSIZE** to adjust ending point
    - Can get overlap of communication/computation in this path, if timing is right
      - Helps to issue MPI_Isend prior to MPI_Irecv
  - R1 – Pipelined **RDMA PUT** method
    - By default, used for messages greater than 512K bytes
    - Use **MPICH_GNI_NDREG_MAXSIZE** to adjust starting point
    - Little chance for communication/computation overlap in this path without using async progress threads

MPI Inter-Node Message type R0

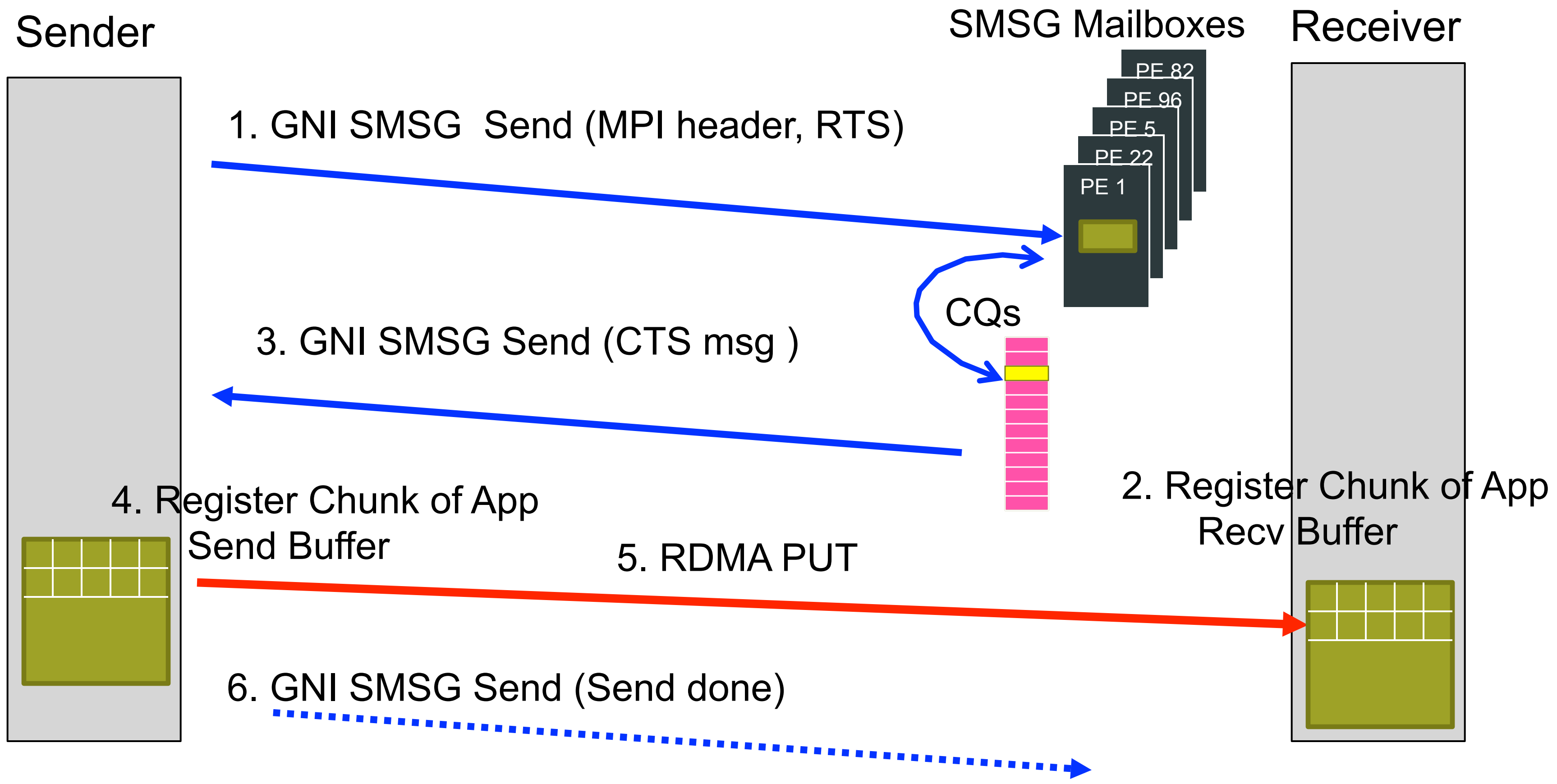
Rendezvous messages using RDMA Get


Sender
SMSG Mailboxes
Receiver
PE 82
PE 96
PE 5
PE 22
PE 1
2. GNI SMSG Send (MPI header)
CQs
1. Register App Send Buffer
3. Register App Recv Buffer
4. RDMA GET
5. GNI SMSG Send (Recv done)

- **No extra data copies**
- **Performance of GET sensitive to relative alignment of send/recv buffers**

MPI Inter-Node Message type R1

Rendezvous messages using RDMA Put


Sender
SMSG Mailboxes
Receiver
PE 82
PE 96
PE 5
PE 22
PE 1
1. GNI SMSG Send (MPI header, RTS)
CQs
3. GNI SMSG Send (CTS msg)
4. Register Chunk of App Send Buffer
2. Register Chunk of App Recv Buffer
5. RDMA PUT
6. GNI SMSG Send (Send done)

- ***Repeat steps 2-6 until all sender data is transferred***
- **Chunksize is MPI_GNI_MAX_NDREG_SIZE (default of 512K bytes)**

Some Useful Environment Variables

MPICH_GNI_MAX_EAGER_MSG_SIZE

- Default is 8192 bytes
- Maximum size message that can go through the eager protocol.
- May help for apps that are sending medium size messages, and do better when loosely coupled. Does application have a large amount of time in MPI_Waitall? Setting this environment variable higher may help.
- Max value is 131072 bytes.
- Remember for this path it helps to pre-post receives if possible.
- Note that a 40-byte message header is included when accounting for the message size.

MPICH_GNI_RDMA_THRESHOLD

- **Controls the crossover point between FMA and BTE path on the Gemini.**
- **Impacts the E1, R0, and R1 paths**
- **If your messages are slightly above or below this threshold, it may benefit to tweak this value.**
  - Higher value: More messages will transfer asynchronously, but at a higher latency.
  - Lower value: More messages will take fast, low-latency path.
- **Default: 1024 bytes**
- **Maximum value is 64K and the step size is 128**
- **All messages using E0 path (GNI Smsg mailbox) will be transferred via FMA regardless of the MPICH_GNI_RDMA_THRESHOLD value**

MPICH_GNI_NUM_BUFS

- Default is 64 32K buffers (2M total)
- Controls number of 32K DMA buffers available for each rank to use in the Eager protocol described earlier
- May help to modestly increase. But other resources constrain the usability of a large number of buffers.

MPICH_GNI_DYNAMIC_CONN

- By default, mailbox connections are established when a rank first sends a message to another rank. This optimizes memory usage for mailboxes. This feature can be disabled by setting this environment variable to *disabled.*

- For applications with all-to-all style messaging patterns, performance may be improved by setting this environment variable to *disabled*.

Questions
?