MPI
for Cray XE/XK7 Systems

Heidi Poxon
Technical Lead & Manager, Performance Tools
Cray Inc.

Agenda

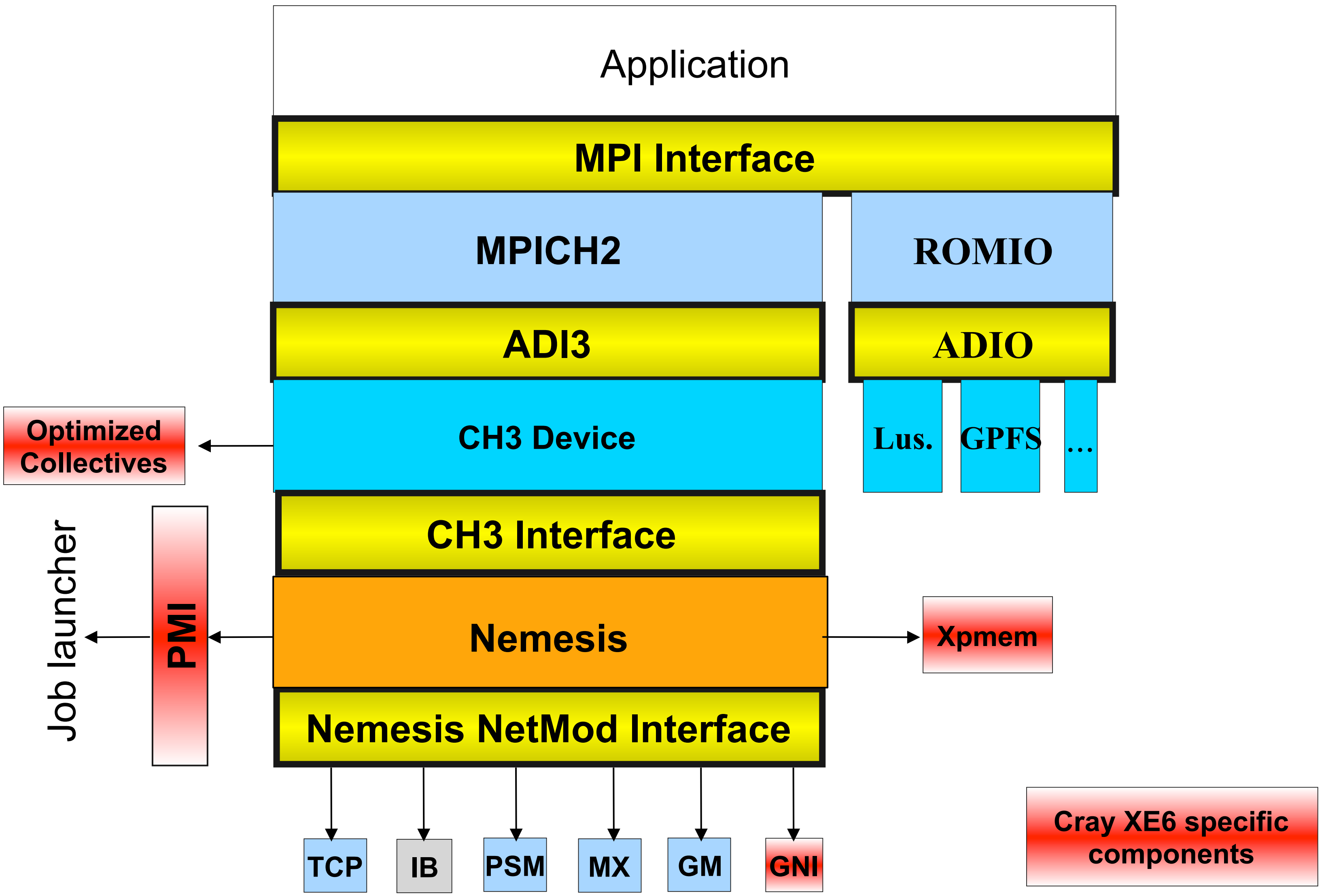
- MPI Overview
- Recent MPI enhancements / differences between ANL MPICH and Cray MPI
- MPI messaging (under the covers)
- Useful environment variables

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 (cray-mpich2/5.6.2.1) last week
- **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.***

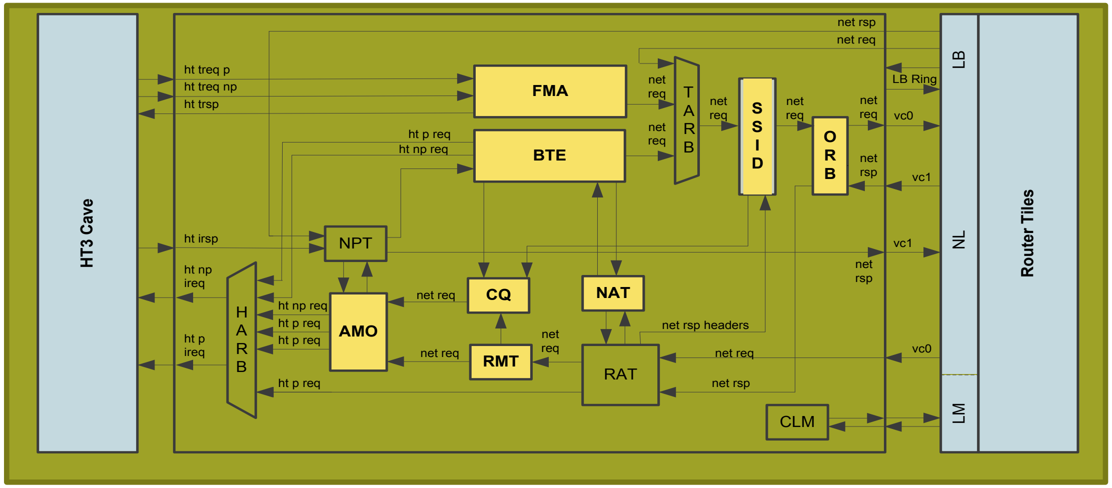
** As of MPT 5.6.0*

MPICH2/Cray layout



Gemini NIC Design

- Fast memory access (FMA)
  - Mechanism for most MPI transfers, involves processor
  - Supports tens of millions of MPI requests per second
- Block transfer engine (BTE)
  - Supports asynchronous block transfers between local and remote memory, in either direction
  - For large MPI transfers that happen in the background
- Hardware pipeline maximizes issue rate
- HyperTransport 3 host interface
- Hardware translation of user ranks and addresses
- AMO cache
- Network bandwidth dynamically shared between NICs



Gemini MPI Features

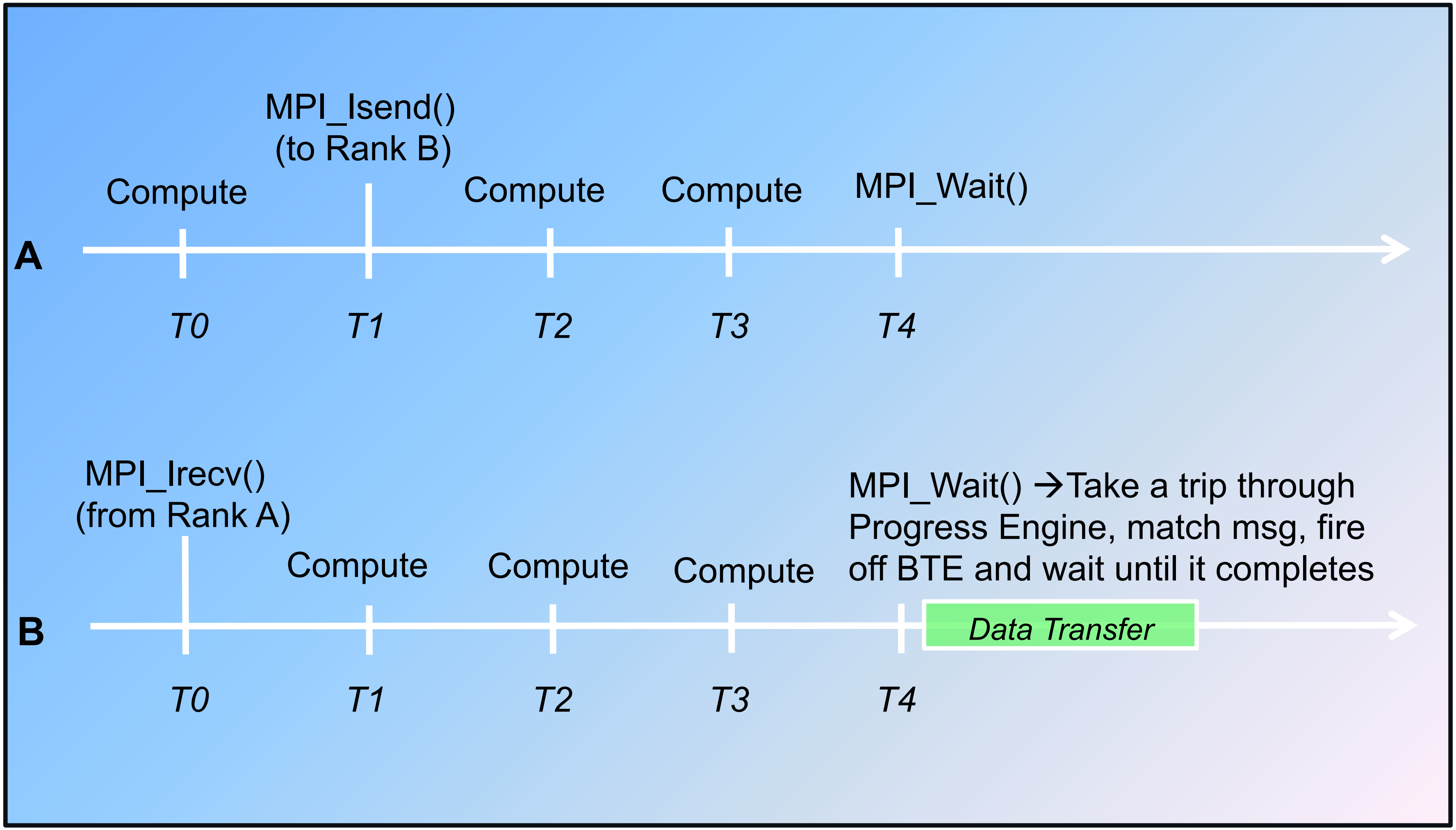
- FMA provides low-overhead OS-bypass
  - Lightweight issue of small transfers
- DMA offload engine
  - Allows large transfers to proceed asynchronously of the application
- Designed to minimize memory usage on large jobs
  - Typically 20MB/process including 4MB buffer for unexpected messages
- AMOs provide a fast synchronization method for collectives
  - AMO=Atomic Memory Operations

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

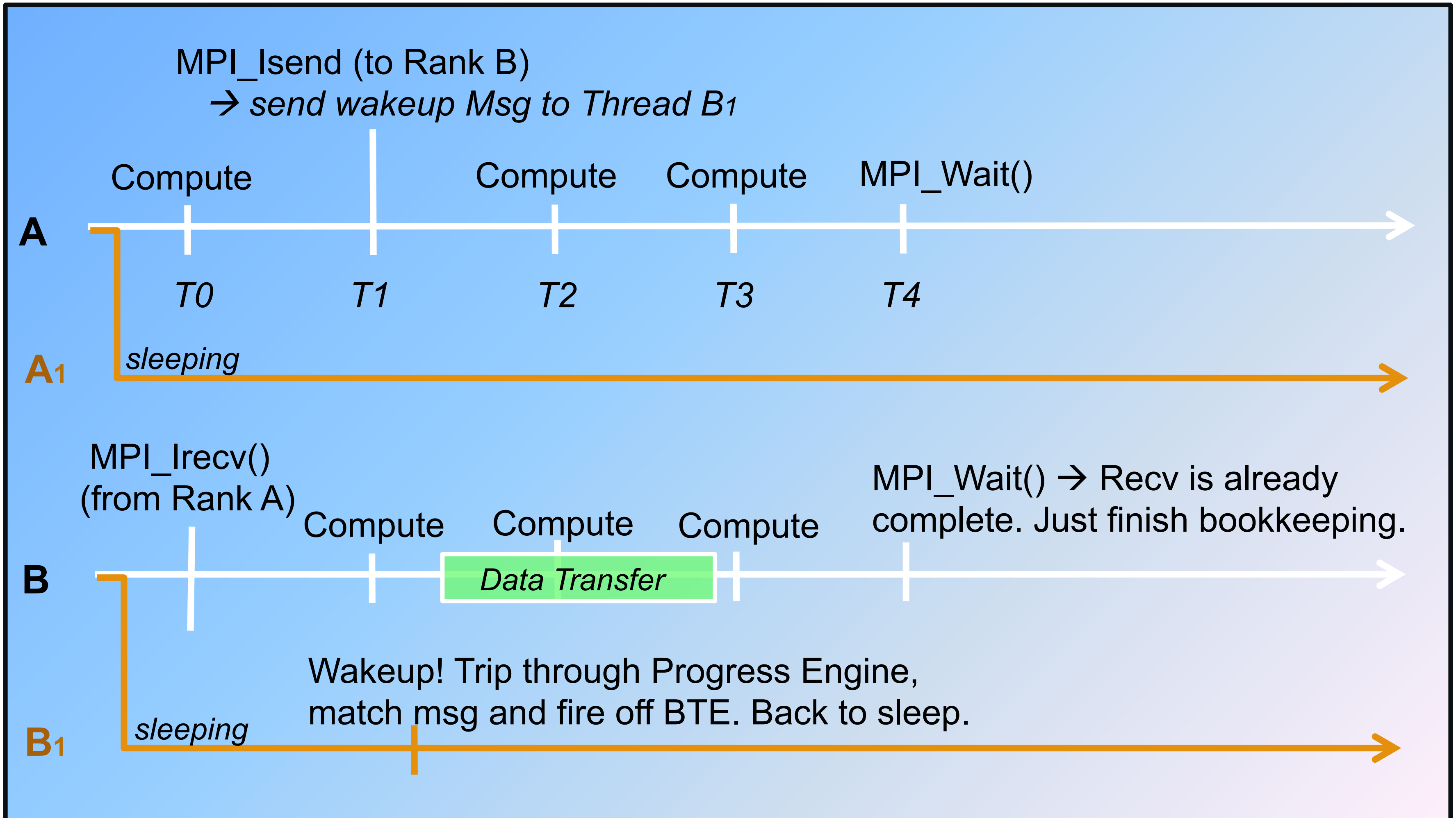
Async Progress Engine Example

2P Example without using Async Progress Threads



Async Progress Engine Example

2P Example using Async Progress Threads



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

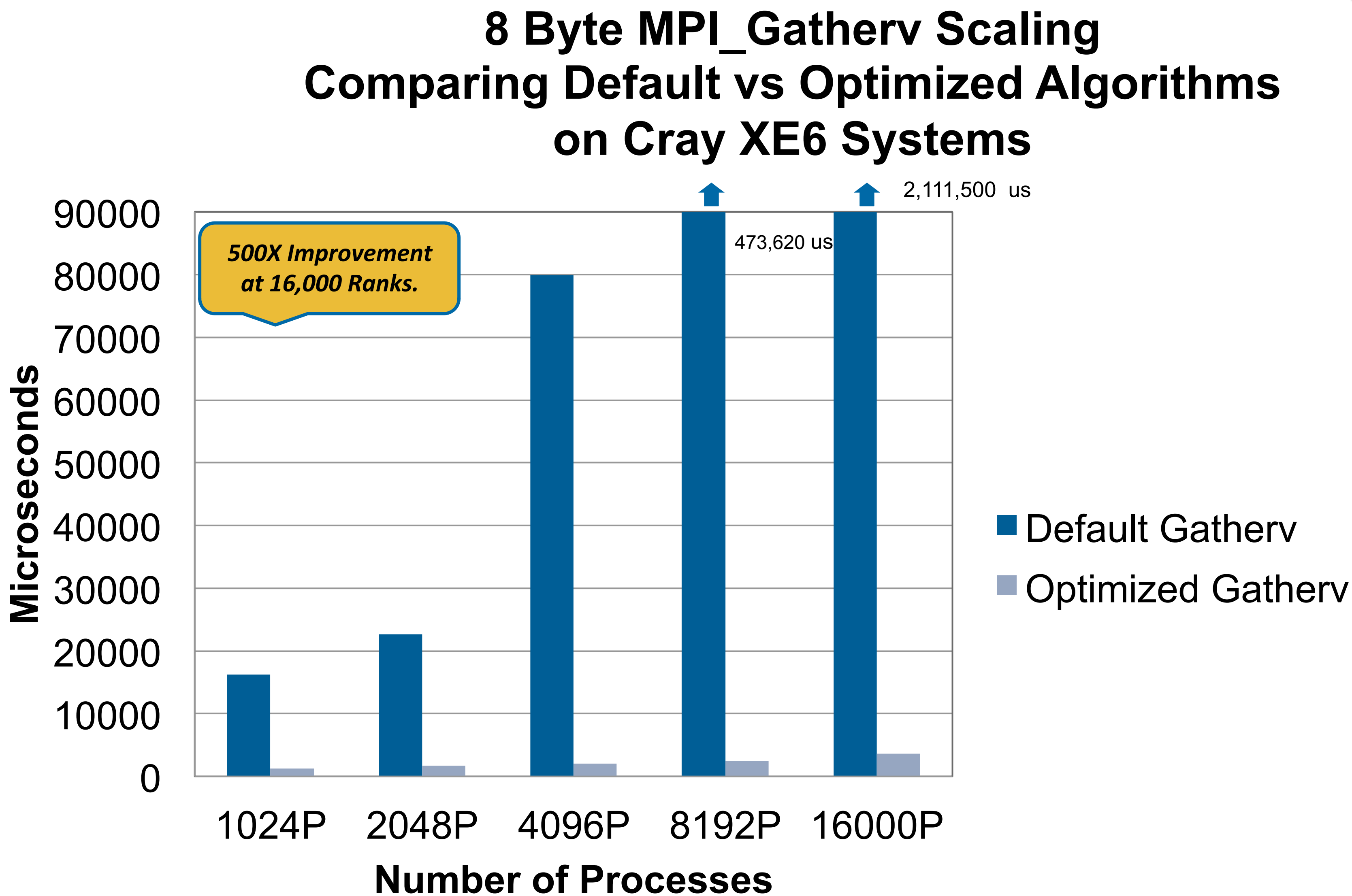
Optimizations on by default unless specified for

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

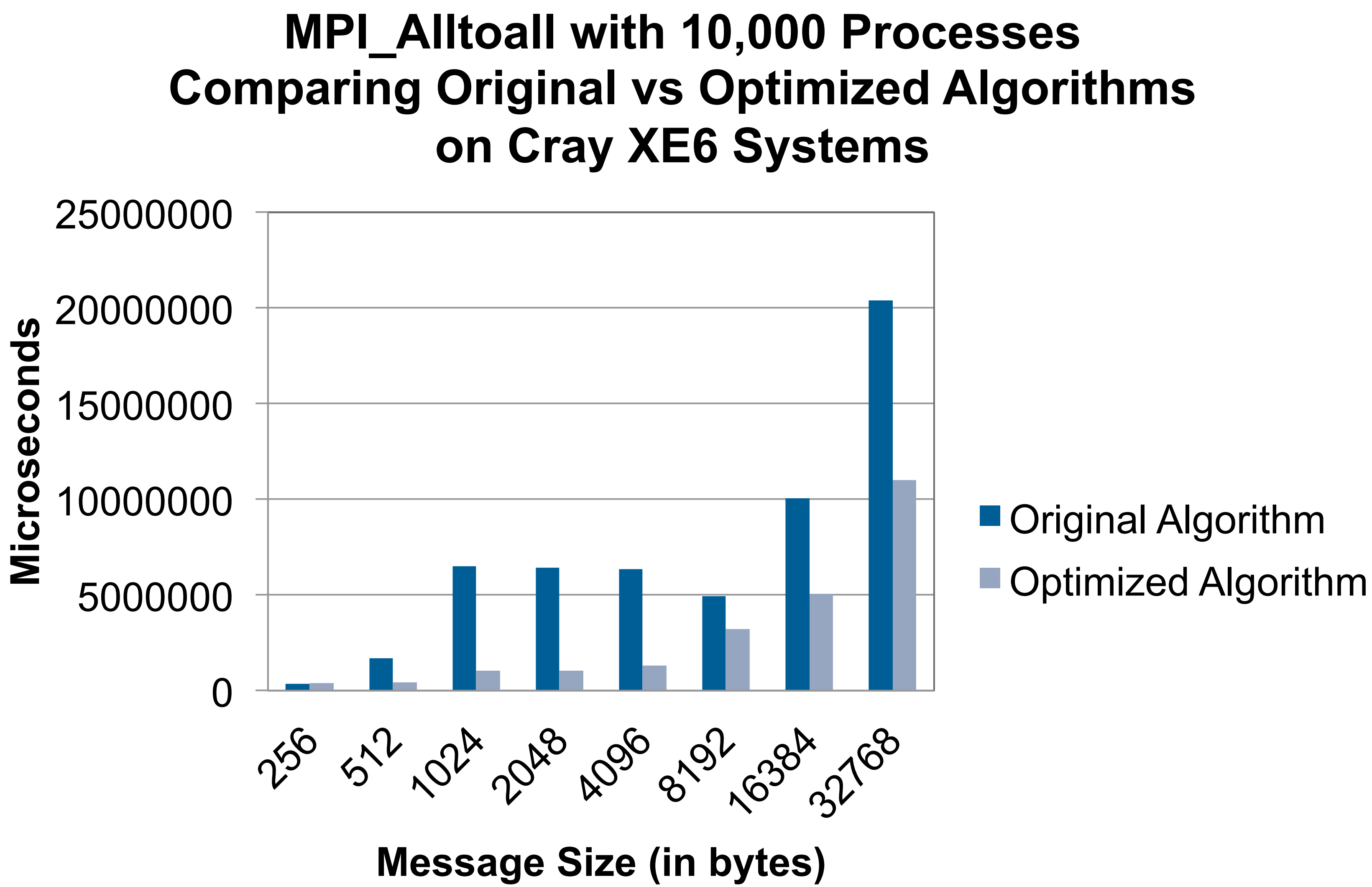
Optimizations off by default unless specified for

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

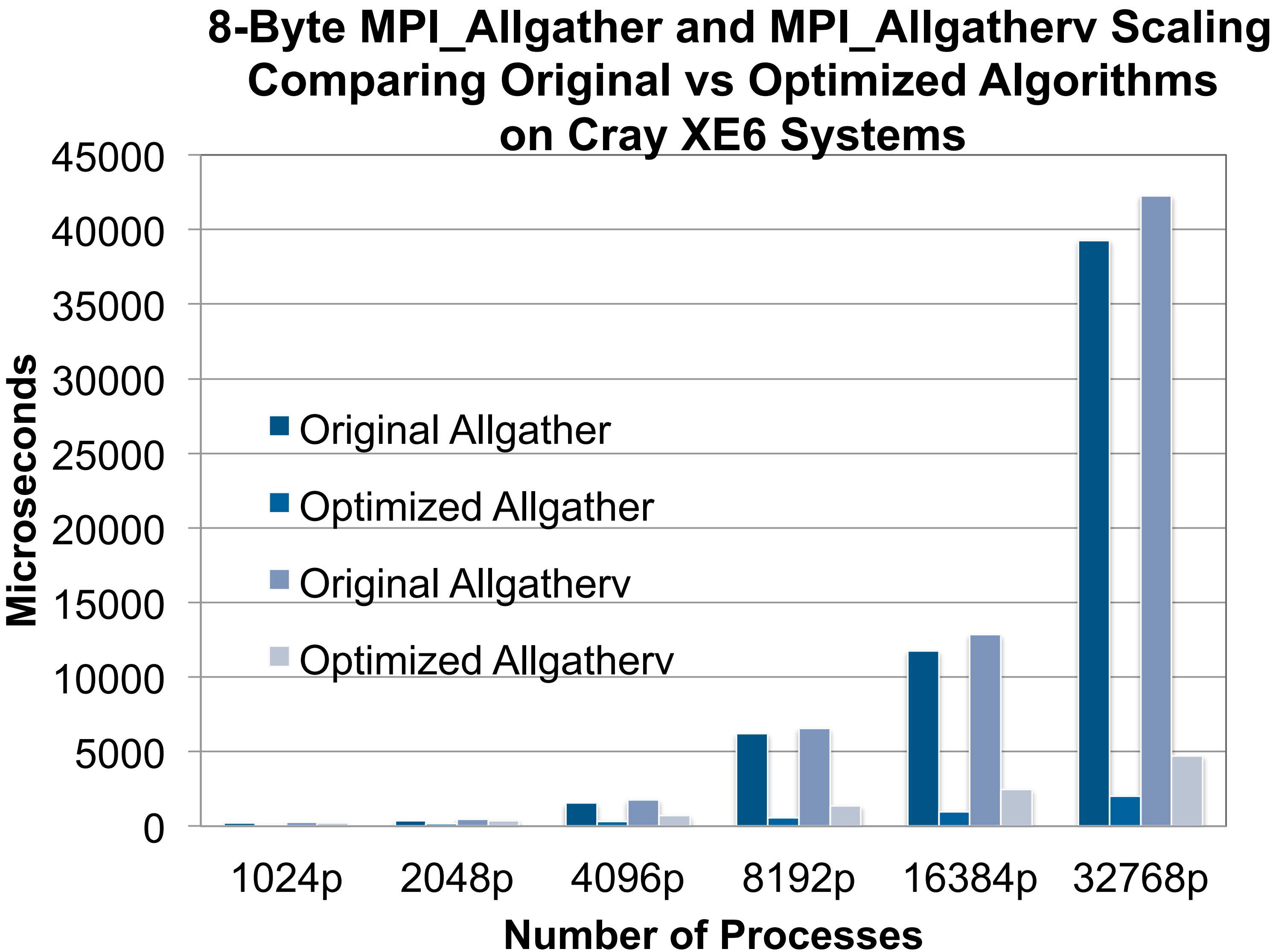
MPI_Gatherv Performance



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
- **Optimize writing and reading GPFS files via DVS with MPI I/O**
  - See the MPICH_MPIIO_DVS_MAXNODES env variable
- **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 describe how to interpret the data and makes suggestions on how to improve your application.

- **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.

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);
}
```

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

MPI Inter-Node Messaging

- **Gemini NIC Resources for Transferring Data**
- **Eager Message Protocol**
  - E0 and E1 Paths
- **Rendezvous Message Protocol**
  - R0 and R1 Paths
- **MPI environment variables that alter those paths**

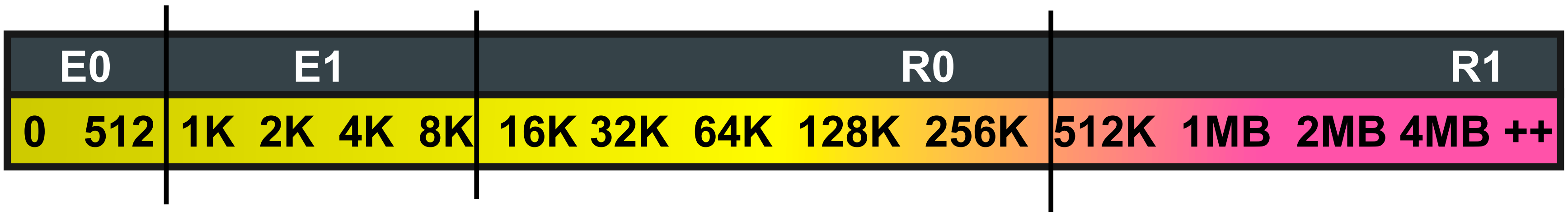
Gemini NIC Resources

- **FMA (Fast Memory Access)**
  - Used for small messages
  - Called directly from user mode
  - Very low overhead → good latency

- **BTE (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

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

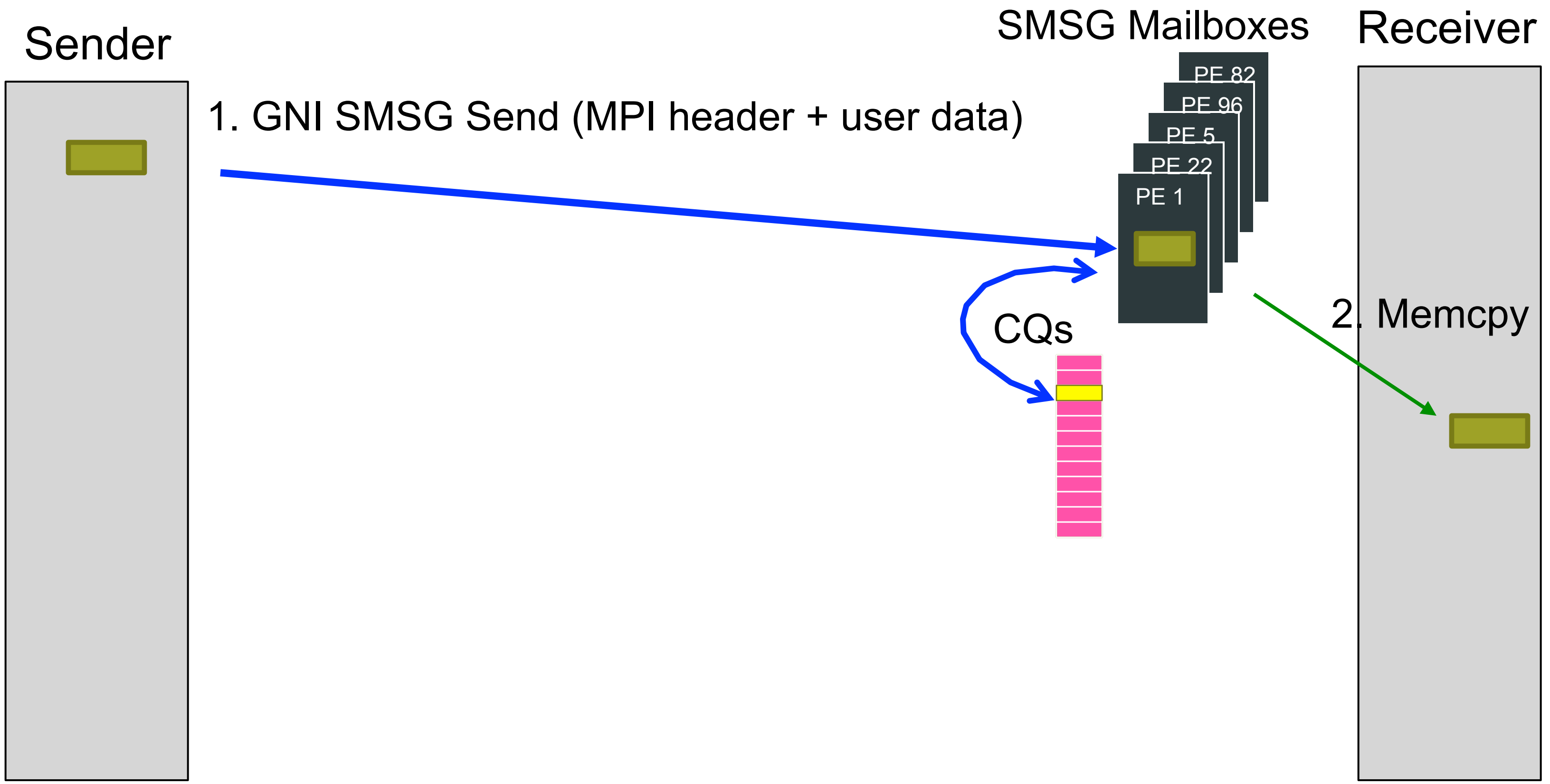
- **Data is transferred when MPI_Send (or variant) encountered**
  - Implies data will be buffered on receiver's node
- **Two EAGER Pathways**
  - **E0** – small messages that fit into GNI SMSG Mailbox
    - Default mailbox size varies with number of ranks in the job

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

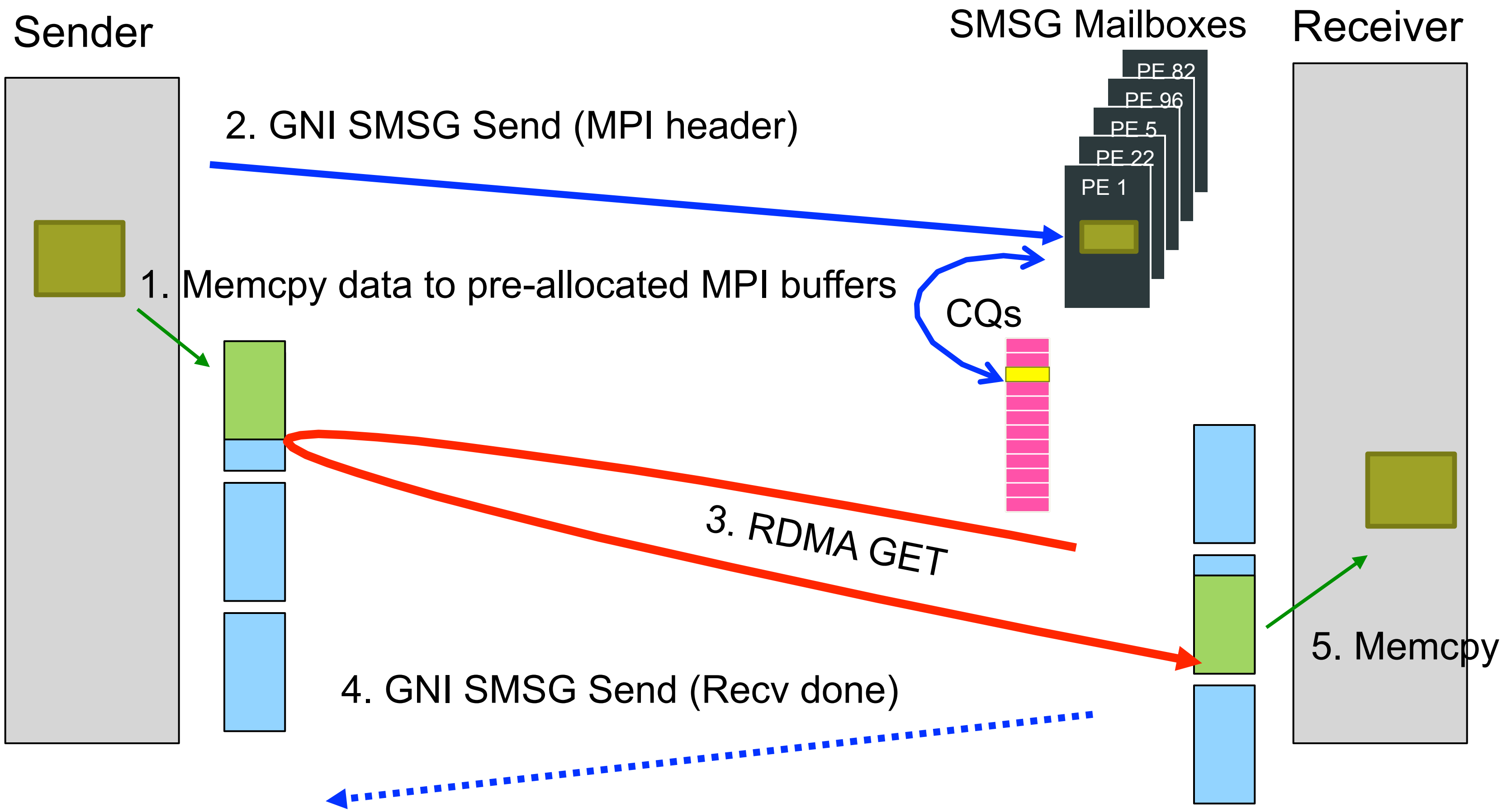
EAGER messages that fit in the GNI SMSG Mailbox



- 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



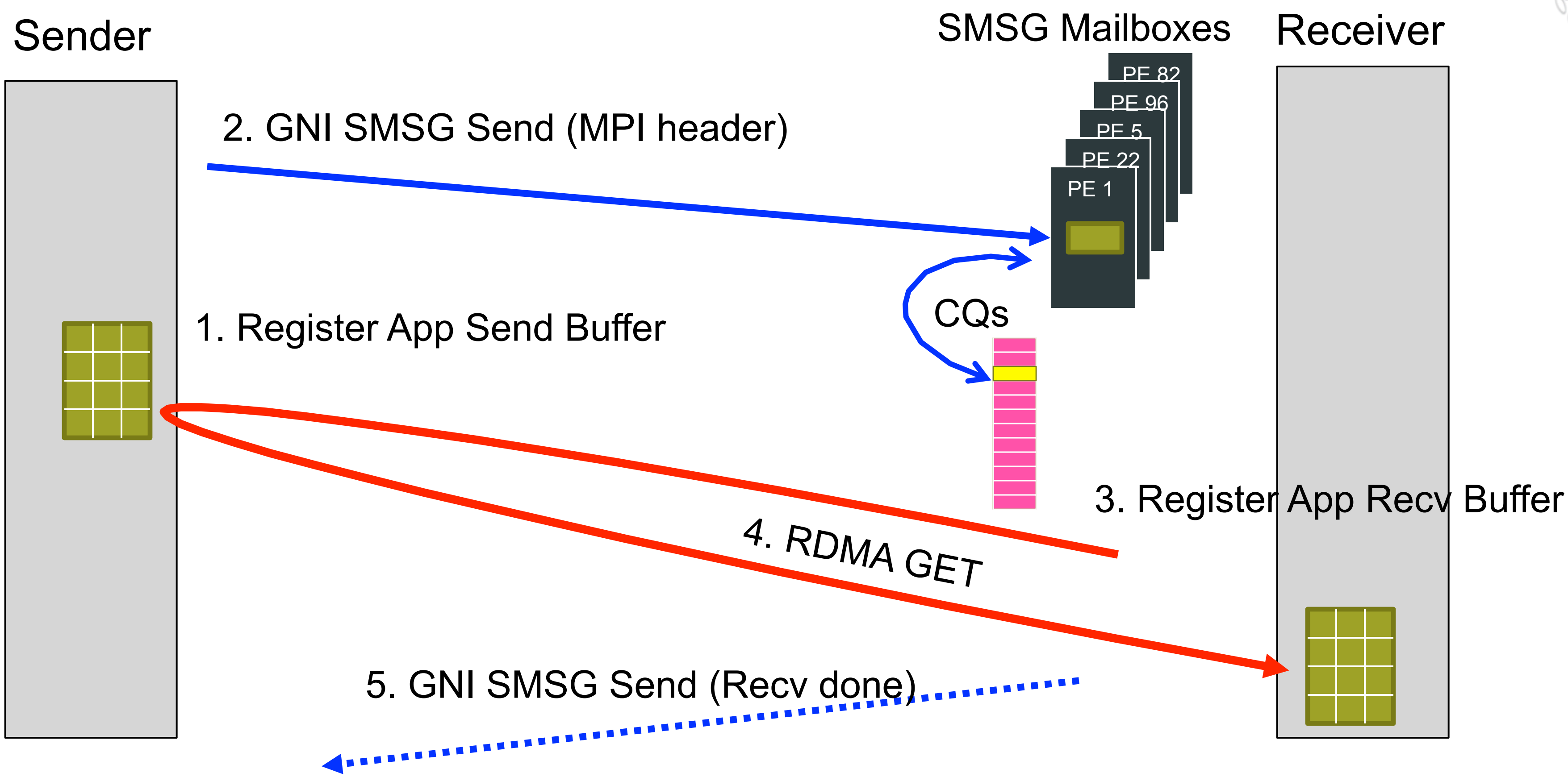
- User data is copied into internal MPI buffers on both send and receive side
- Default MPICH_GNI_NUM_BUFS is 64 (each buffer is 32K)
- Internal MPI buffers use large pages

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 <u>prior</u> 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

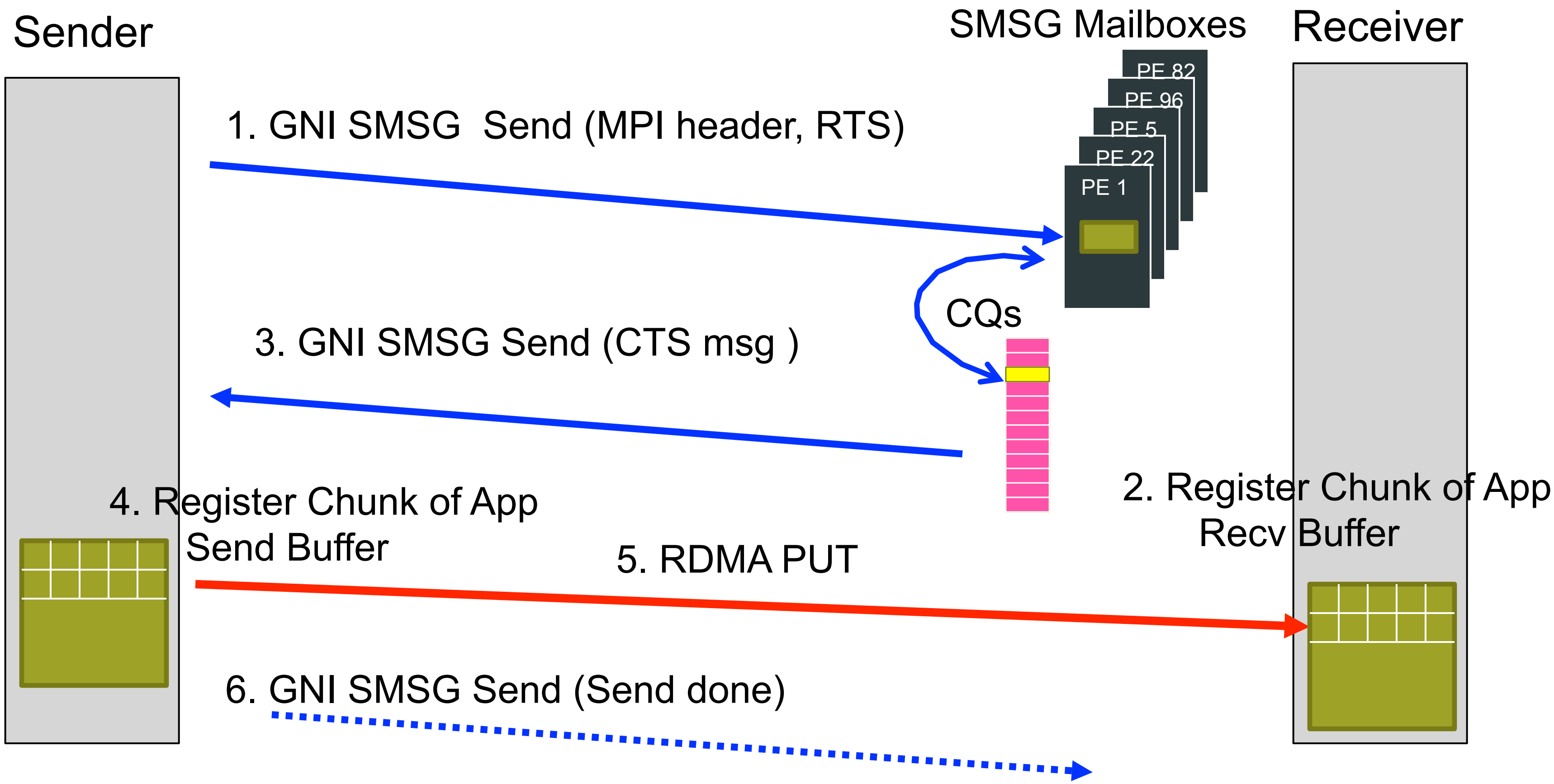
Rendezvous messages using RDMA Get



- **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



- ***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 ?