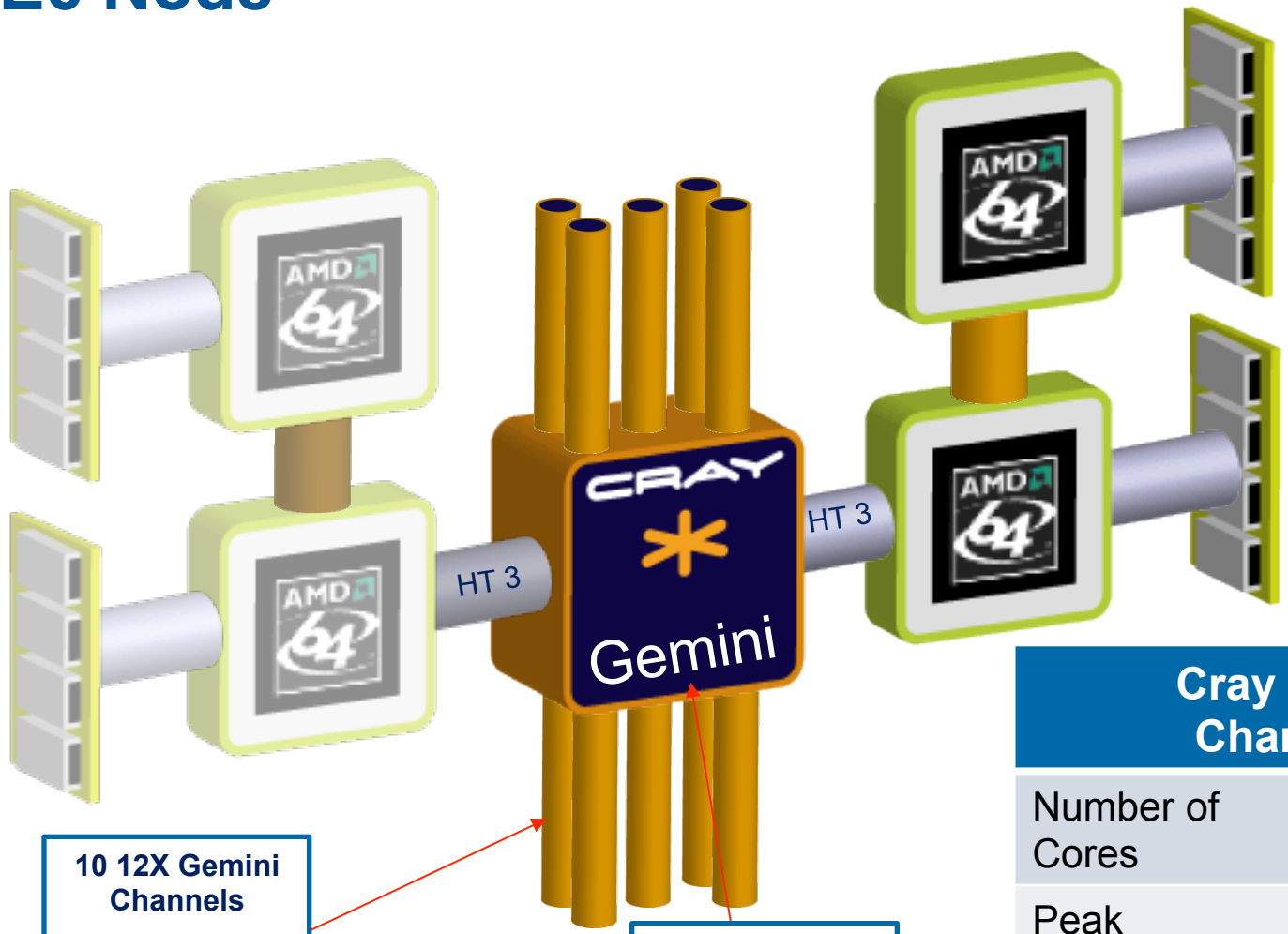
Programming Models For Cray XE Systems

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

Agenda

- Cray programming models on Interlagos
- OpenMP
- PGAS (UPC, Fortran coarrays)
- MPI

XE6 Node



10 12X Gemini Channels

(Each Gemini acts like two nodes on the 3D Torus)

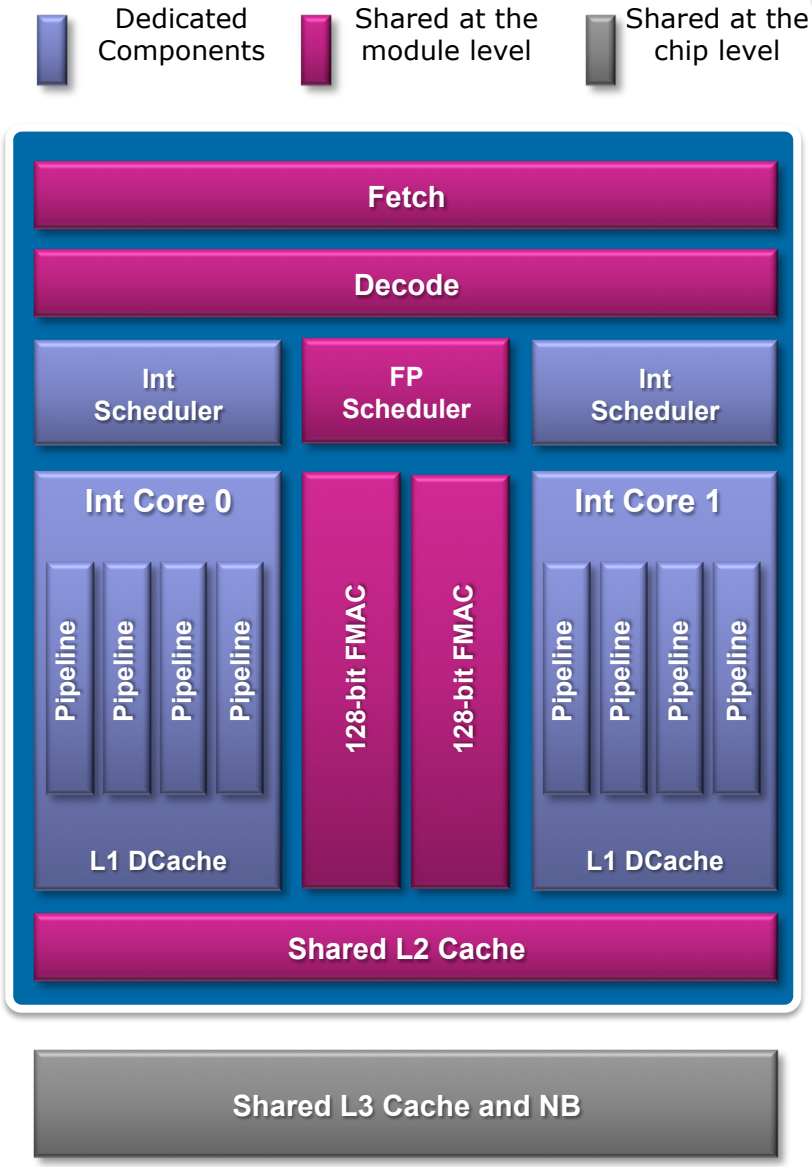
High Radix YARC Router with adaptive Routing

168 GB/sec capacity

Cray Baker Node Characteristics	
Number of Cores	32*
Peak Performance	~300 Gflops/s
Memory Size	64 GB per node
Memory Bandwidth	85 GB/sec

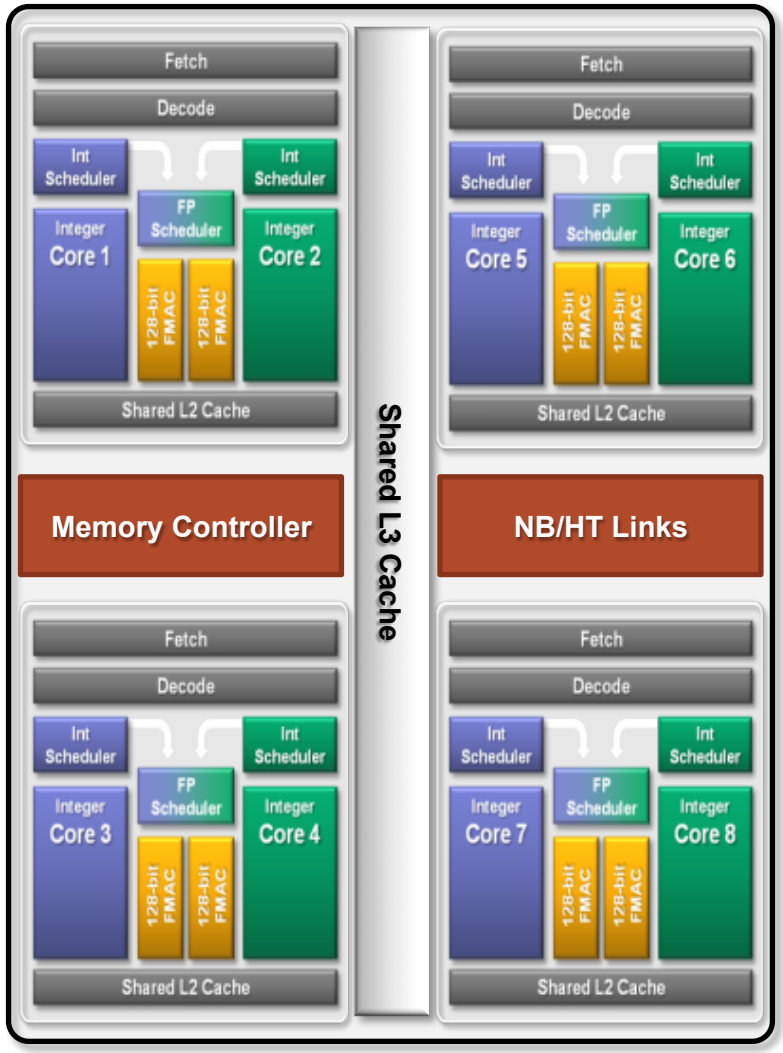
Interlagos Processor Architecture

- **Interlagos is composed of a number of "Bulldozer modules" or "Compute Unit"**
  - A compute unit has shared and dedicated components
    - There are two independent integer units; shared L2 cache, instruction fetch, Icache; and a *shared*, 256-bit Floating Point resource
  - A single Integer unit can make use of the entire Floating Point resource with 256-bit AVX instructions
    - Vector Length
      - 32 bit operands, VL = 8
      - 64 bit operands, VL = 4



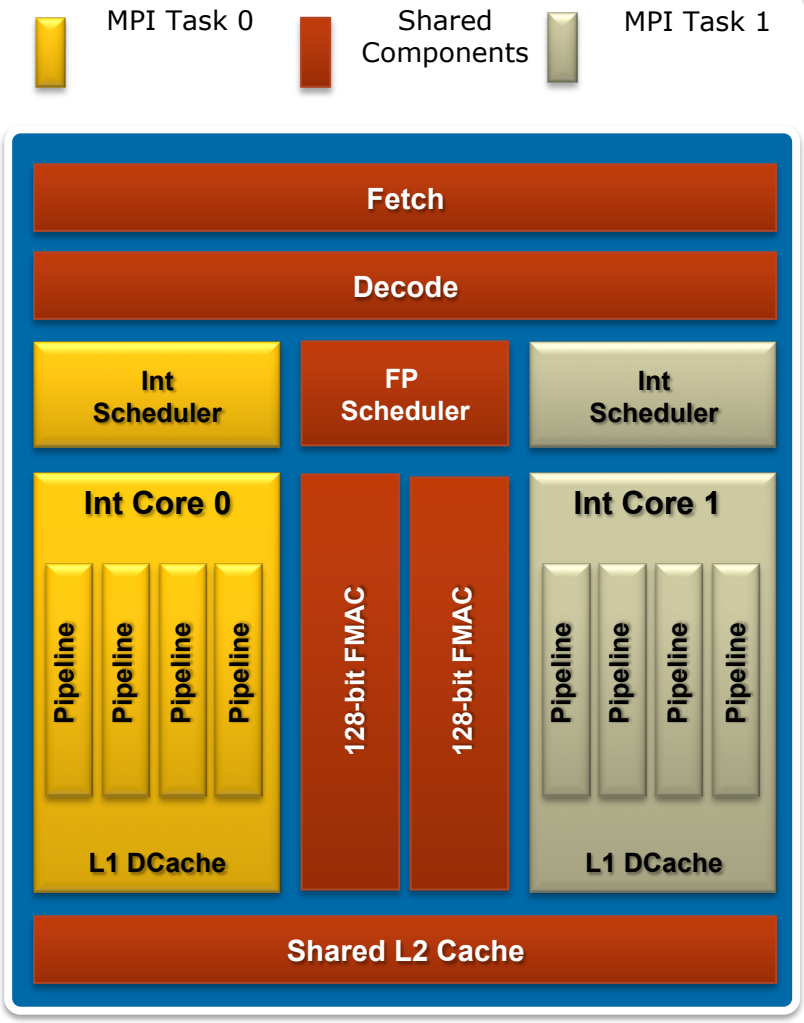
Building an Interlagos Processor

- **Each processor die is composed of 4 compute units**
  - The 4 compute units share a memory controller and 8MB L3 data cache
    - Each processor die is configured with two DDR3 memory channels and multiple HT3 links



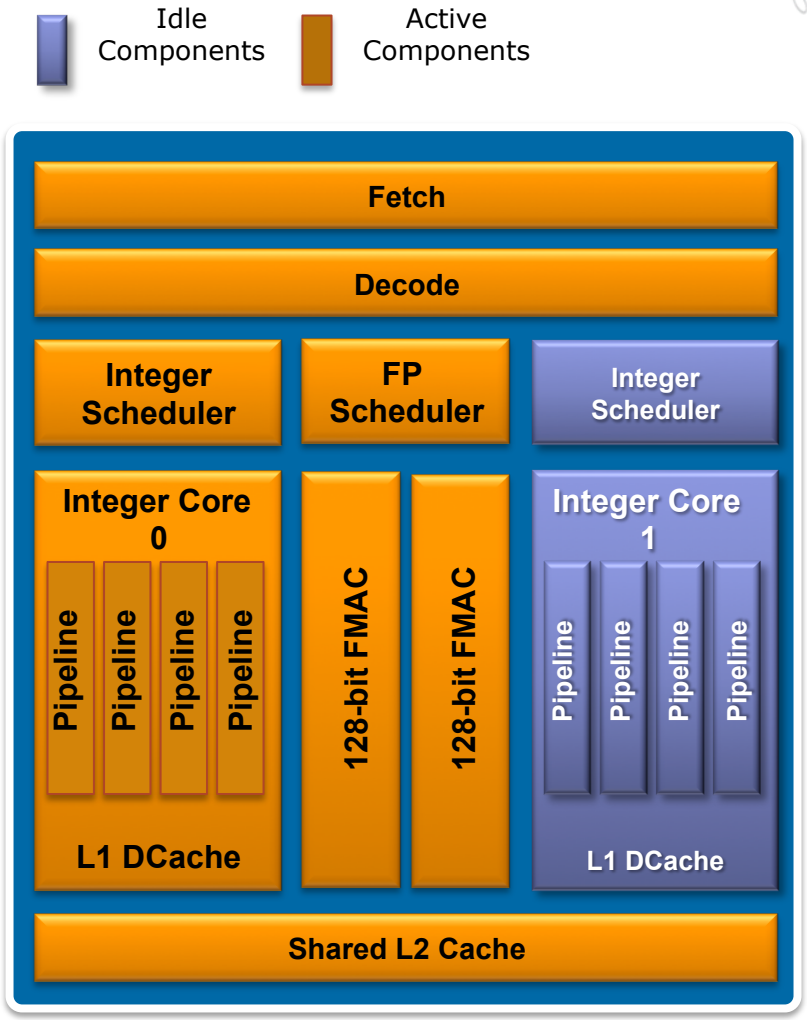
Two MPI Tasks on a Compute Unit ("Dual-Stream Mode")

- **An MPI task is pinned to each integer unit**
  - Each integer unit has exclusive access to an integer scheduler, integer pipelines and L1 Dcache
  - The 256-bit FP unit, instruction fetch, and the L2 Cache are shared between the two integer units
    - 256-bit AVX instructions are dynamically executed as two 128-bit instructions if the 2nd FP unit is busy
- **When to use**
  - Code is highly scalable to a large number of MPI ranks
  - Code can run with a 2GB per task memory footprint
  - Code is not well vectorized



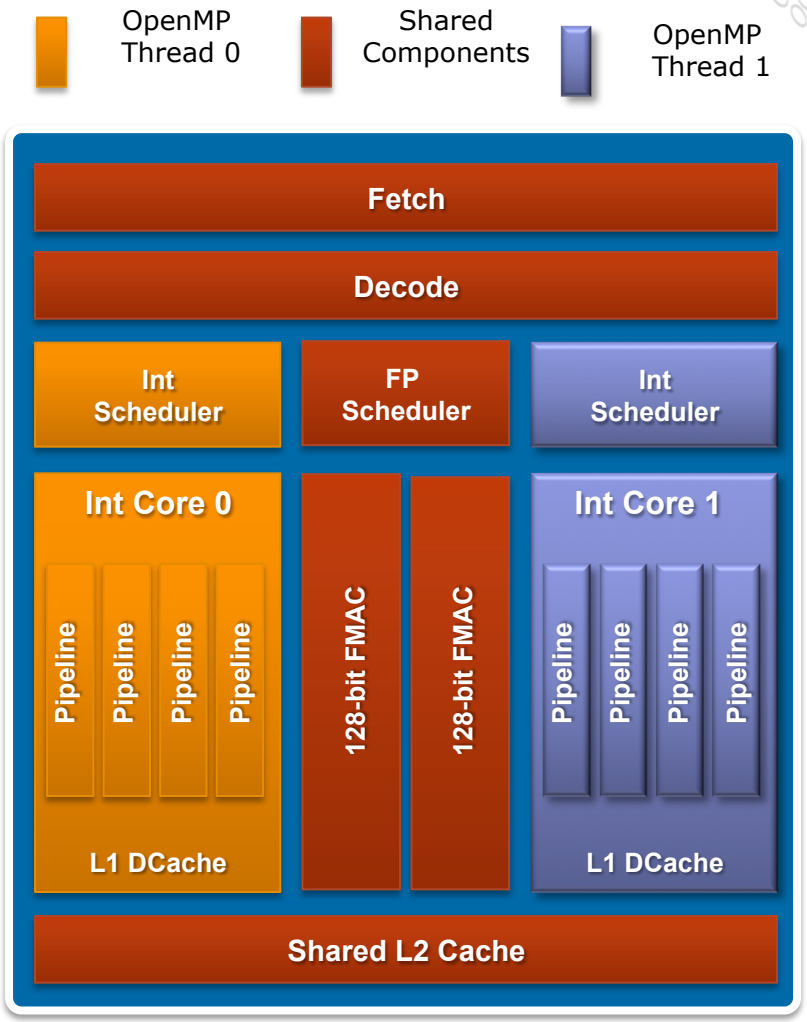
One MPI Task on a Compute Unit ("Single Stream Mode")

- **Only one integer unit is used per compute unit**
  - This unit has exclusive access to the 256-bit FP unit and is capable of 8 FP results per clock cycle
  - The unit has twice the memory capacity and memory bandwidth in this mode
  - The L2 cache is effectively twice as large
  - The peak of the chip is not reduced
- **When to use**
  - Code is highly vectorized and makes use of AVX instructions
  - Code benefits from higher per task memory size and bandwidth



One MPI Task per compute unit with Two OpenMP Threads ("Dual-Stream Mode")

- **An MPI task is pinned to a compute unit**
- **OpenMP is used to run a thread on each integer unit**
  - Each OpenMP thread has exclusive access to an integer scheduler, integer pipelines and L1 Dcache
  - The 256-bit FP unit and the L2 Cache is shared between the two threads
  - 256-bit AVX instructions are dynamically executed as two 128-bit instructions if the 2nd FP unit is busy
- **When to use**
  - Code needs a large amount of memory per MPI rank
  - Code has OpenMP parallelism at each MPI rank



Running in Dual or Single-Stream modes

- **Dual-Stream mode is** the current **default** mode. General use does not require any options. CPU affinity is set automatically by ALPS.
- **Single-Stream mode** is **specified through** the **`-j` aprun option**. Specifying -j 1 tells aprun to place 1 process or thread on each compute unit.
- When OpenMP threads are used, the `-d` option must be used to specify how many threads will be spawned per MPI process. See the `aprun(1)` man page for more details. The `aprun -N` option may be used to specify the number of MPI processes to assign per compute node or `-S` to specify the number of processes per Interlagos die. Also, the environment variable `$OMP_NUM_THREADS` needs to be set to the correct number of threads per process.
- For example, the following spawns 4 MPI processes, each with 8 threads, using 1 thread per compute unit.

```
OMP_NUM_THREADS=8  aprun -n 4 -d 8 -j 1 ./a.out
```

NUMA Considerations

- Each Interlagos processor has 2 NUMA memory domains, each with 4 Bulldozer Modules. Access to memory located in a remote NUMA domain is slower than access to local memory. Bandwidth is lower, and latency is higher.
- OpenMP performance is usually better when all threads in a process execute in the same NUMA domain. For the Dual-Stream case, 8 CPUs share a NUMA domain, while in Single-Stream mode 4 CPUs share a NUMA domain. Using a larger number of OpenMP threads per MPI process than these values may result in lower performance due to cross-domain memory access.
- When running 1 process with threads over the NUMA domains, it's critical to initialize (not just allocate) memory from the thread that will use it in order to avoid NUMA side effects.

OpenMP

- **Supported in the Cray Compiling Environment (CCE)**
  - Latest release: CCE 8.1.4 (8.1.5 available February 28)
  - OpenMP directives recognized by default with CCE (no need for command-line option)
  - **OpenMP 3.1 compliant**, working on OpenMP 4.0 compliance
- **Participant in OpenMP standard committee**

- **OpenMP and automatic multithreading fully integrated**
  - Share the same runtime and resource pool
  - Aggressive loop restructuring and scalar optimization done in the presence of OpenMP
  - Consistent interface for managing OpenMP and automatic multithreading

Adding OpenMP to an MPI Program

- **For the next decade (at least) all HPC systems will have the same basic architecture:**
  - Message passing between nodes
  - Multithreading within the node
    - MPI will not do
  - Vectorization at the lowest level
    - SSE, AVX...
    - GPU, MIC...

- **Current petascale applications are not structured to take advantage of these architectures**
  - Currently 80-90% of applications use a single level of parallelism
    - message passing between cores of the MPP system
  - Looking forward, application developers are faced with a significant task in preparing their applications for the future

Three Levels of Parallelism Required

1. Developers will continue to use MPI between nodes or sockets

2. Developers must address using a shared memory programming paradigm on the node

3. Developers must vectorize low level looping structures

While there is a potential acceptance of new languages for addressing all levels directly. Most developers cannot afford this approach until they are assured that the new language will be accepted and the generated code is within a reasonable performance range

PGAS for Cray XE Systems

PGAS

- **PGAS languages (UPC & Fortran Coarrays) fully optimized and integrated into the compiler**
  - UPC 1.2 and Fortran 2008 coarray support
  - No preprocessor involved
  - Target the network appropriately
  - Full debugger support with Allinea's DDT

Enhancements / Changes in CCE 8.1 (3Q 2012)

- **Full support for the Fortran 2008 language standard**
  - 'ftn –h caf' option to recognize Fotran coarray syntax enabled by default
  - Predefined macro _CRAY_COARRAY defined with –h caf
- **UPC is still an extension to C (not part of the language standard) so it is NOT enabled by default**
- **Additional UPC and Fortran coarray shared data structures are now automatically grouped into block transfers, yielding improved data transfer speeds**

Enhancements / Changes in CCE 8.1 (3Q 2012)

- Improvement to conflict detection for writes to UPC shared data, or to Fortran coarrays, allows applications to achieve faster message rates for the common case of no write conflicts
- '**cc –h bounds**' option provides checking of UPC shared array accesses to ensure that are within acceptable boundaries (use –h nobounds to disables these checks)
- Some system headers were removed from upc.h, including stdlib.h which includes sys/types.h
  - Do not rely on upc.h to include other system headers
  - For example, include stdint.h in your code if you need sys/types.h

Enhancements for CCE 8.2

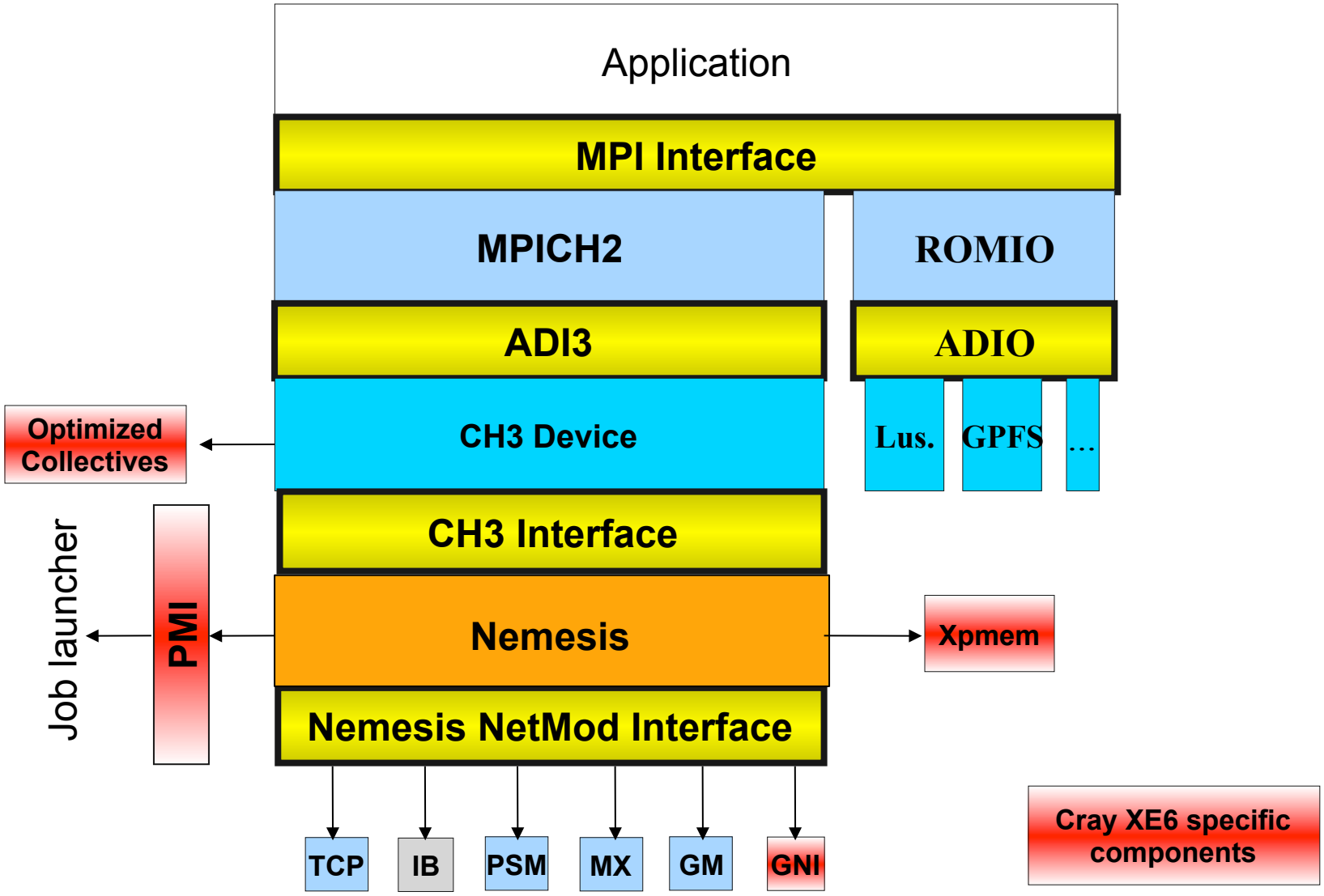
- **Targeted release date: 3Q 2013**
- **The following are planned enhancements**
  - Support for UPC 1.3 specification
  - Add UPC barrier and all-reduce collectives with PE subset support
  - Provide application filename and line number information for PGAS runtime errors
  - Introductory version of Coarry C++, a data distribution model for C++, similar to Fortran Coarrays
    - Looking for “friendly users” to try a prototype of Coarray C++ to give us feedback

MPI for Cray XE/XK7 Systems

Cray MPI Overview

- MPT 5.6.0 released November 2012
- MPT 5.6.2 coming in February 2013
- 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.*

MPICH2/Cray layout



Gemini Features Used by MPI

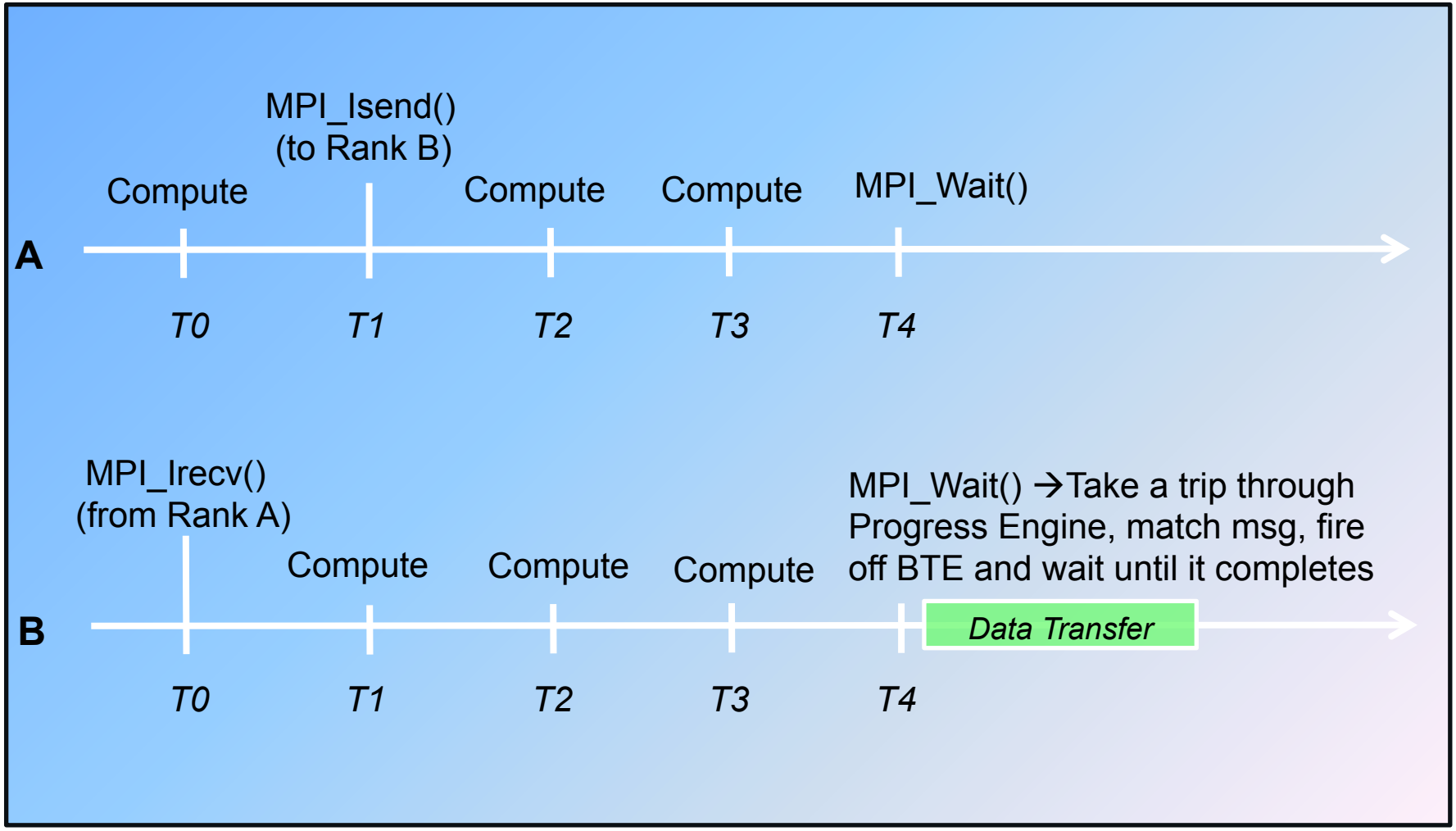
- **FMA (Fast Memory Access)**
  - Used for small messages
  - Called directly from user mode
  - Very low overhead ➔ good latency
- **DMA offload engine (BTE or Block Transfer Engine)**
  - Used for larger messages
  - All ranks on node share BTE resources (4 virtual channels / node)
  - Processed via 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

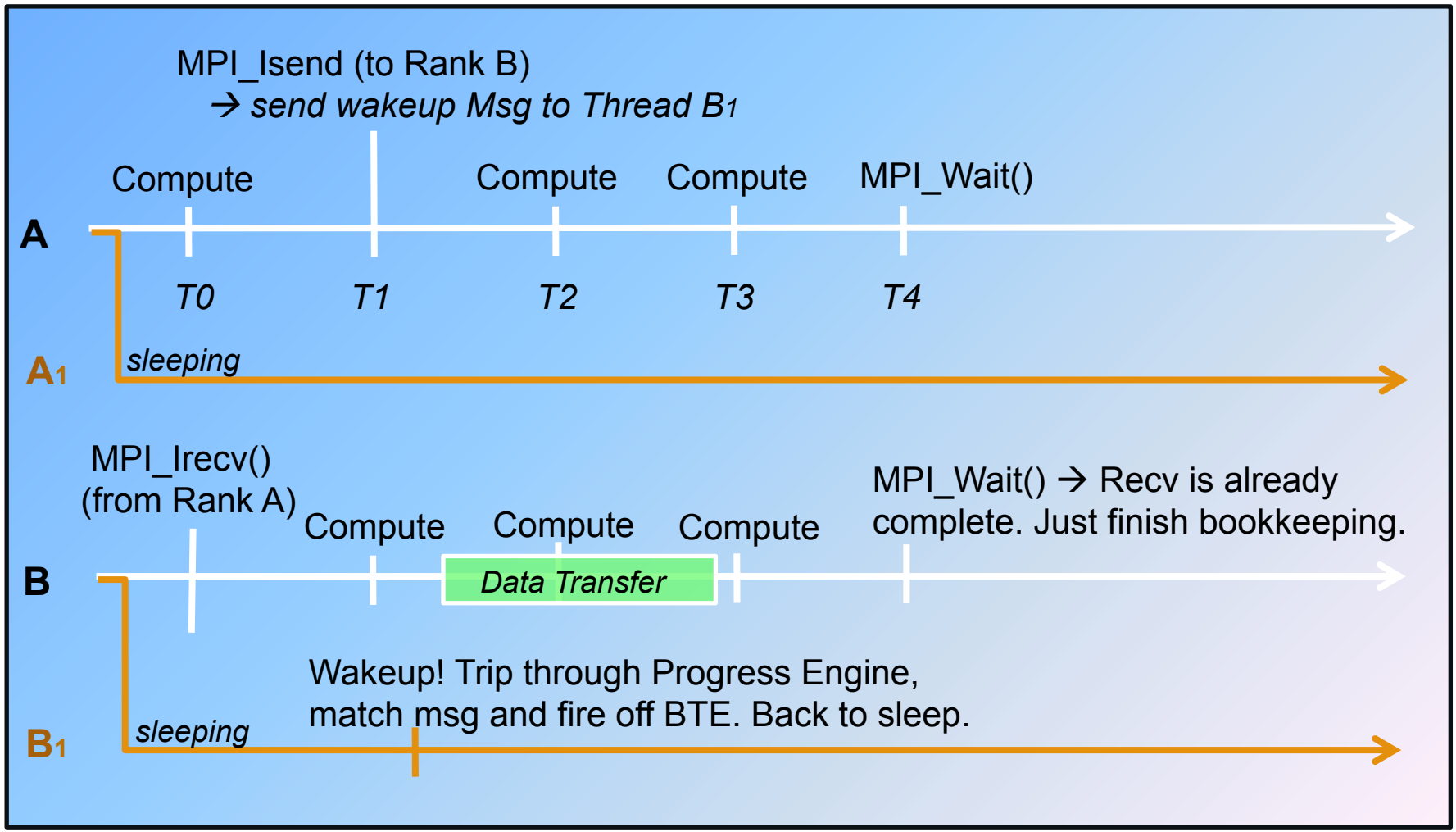
Async Progress Engine Example

2P Example without using Async Progress Threads



Async Progress Engine Example

2P Example using Async Progress Threads



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

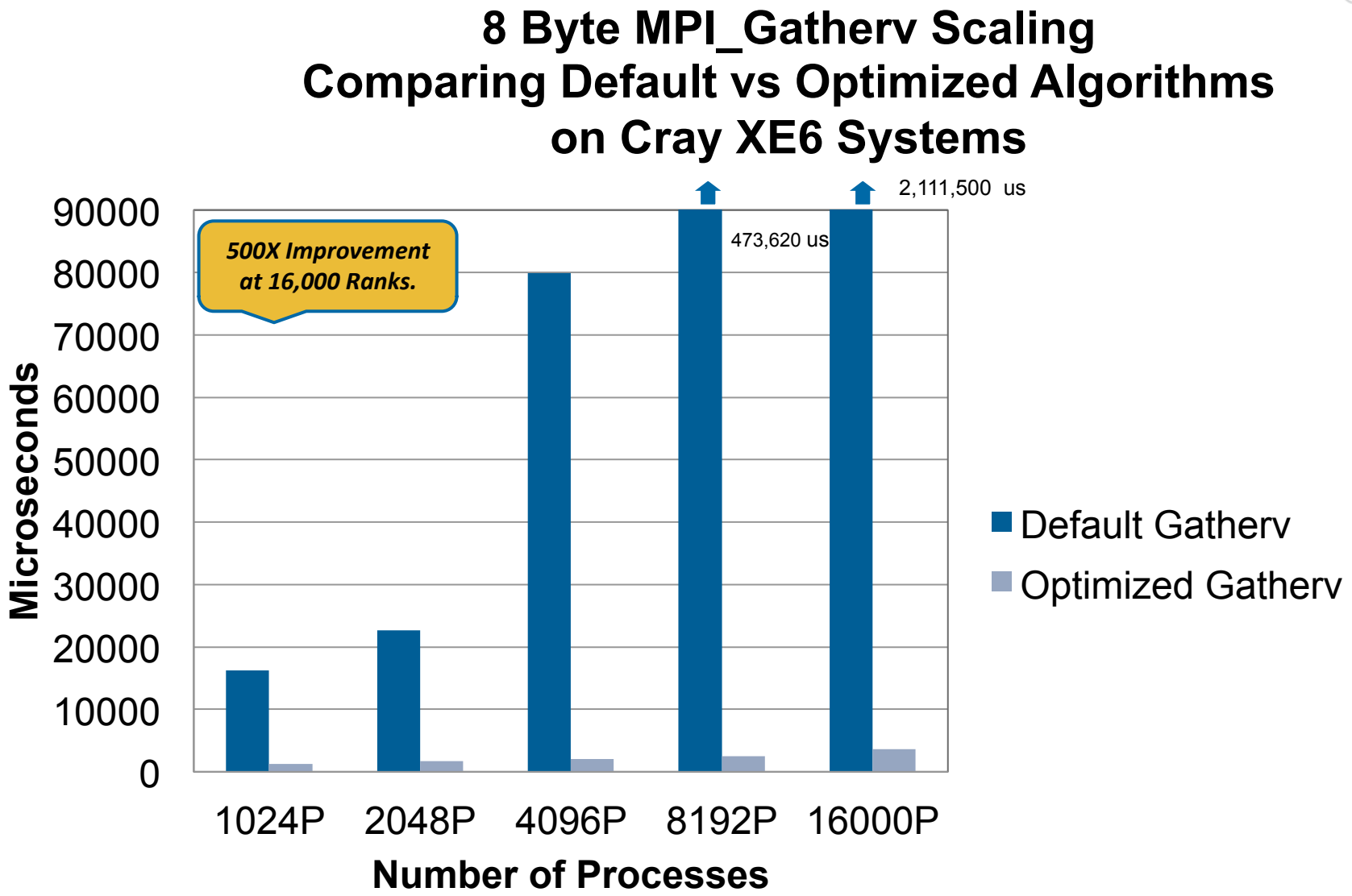
Optimizations on by default unless specified for:

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

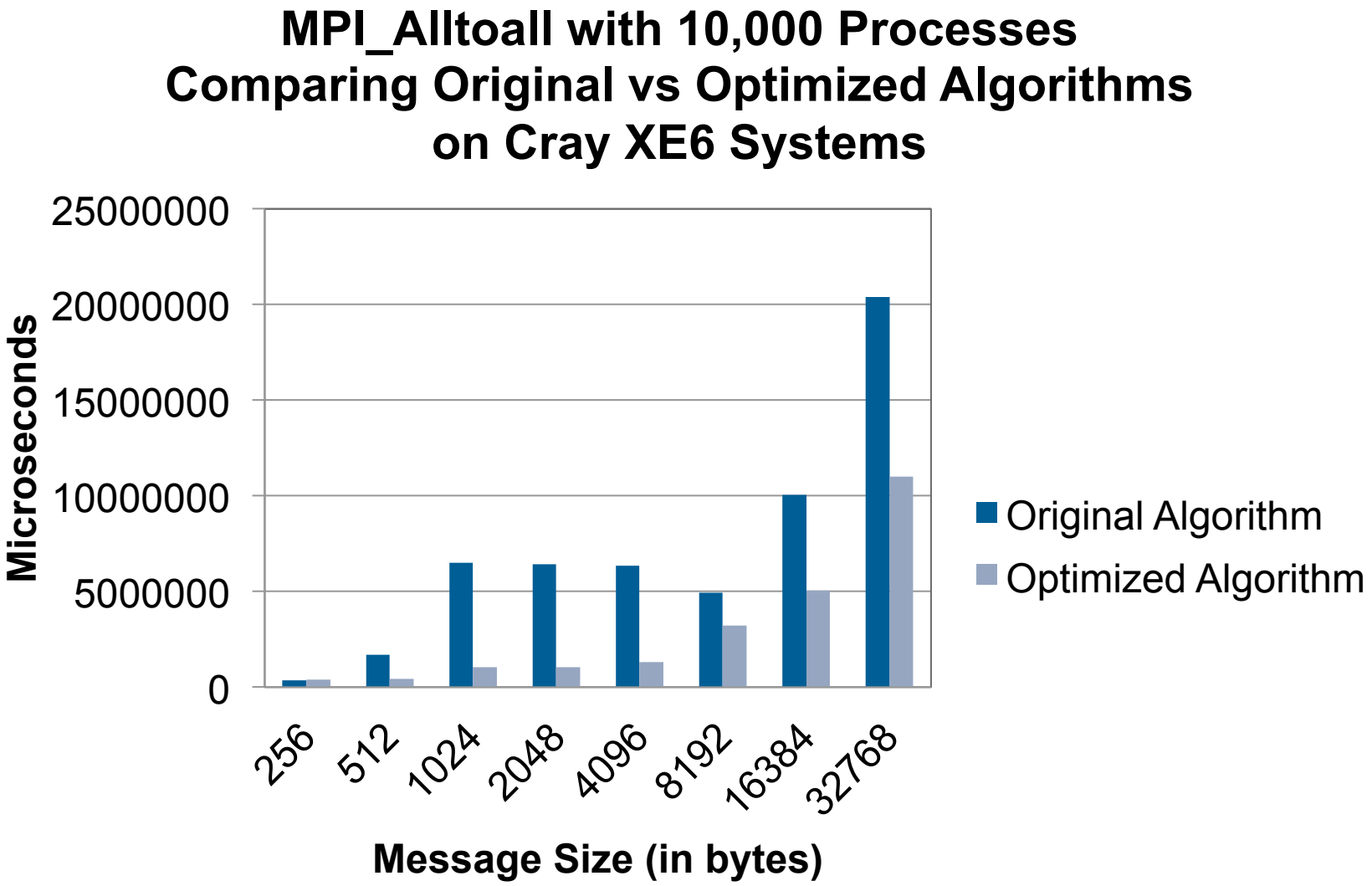
Optimizations off by default unless specified for

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

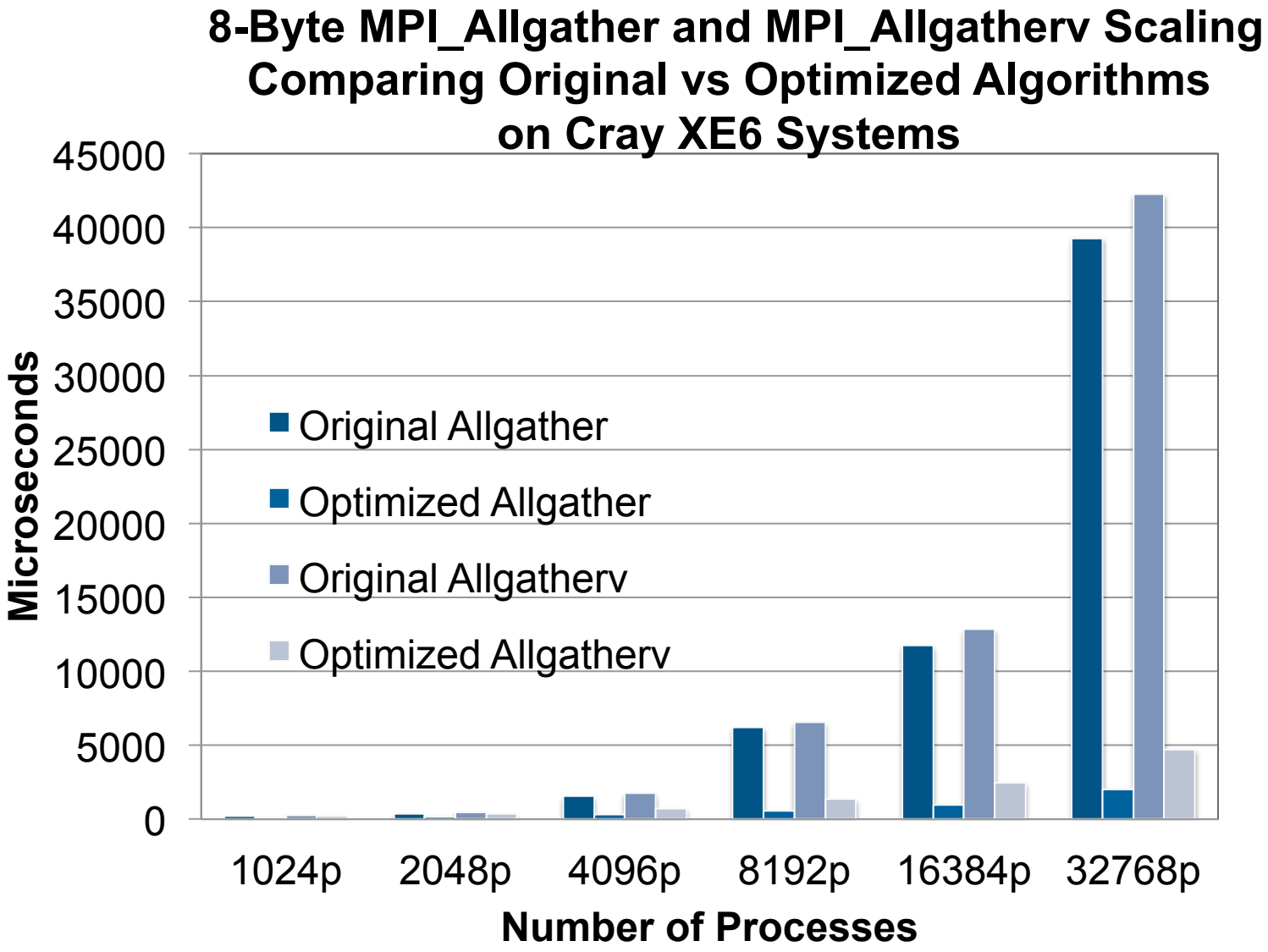
MPI_Gatherv Performance



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 Order	On-Node Bytes/PE	On-Node Bytes/PE% of Total Bytes/PE	MPICH_RANK_REORDER_METHOD
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_METHOD to 3 prior to
# executing the program.
#
0,532,64,564,32,572,96,540,8,596,72,524,40,604,24,588
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
73,395,81,427,57,459,17,419,113,491,49,387,89,451,121,483
6,436,102,468,70,404,38,412,14,444,46,476,110,508,78,500
86,396,30,428,62,460,54,492,118,420,22,452,94,388,126,484
129,563,193,531,161,571,225,539,241,595,233,523,249,603,185,555
153,587,169,627,137,635,201,619,177,515,145,579,209,547,217,611
7,405,71,469,39,437,103,413,47,445,15,509,79,477,31,501
111,397,63,461,55,429,87,421,23,493,119,389,95,453,127,485
134,402,198,434,166,410,230,442,238,466,174,506,158,394,246,474
190,498,254,426,142,458,150,386,182,418,206,490,214,450,222,482
128,533,192,541,160,565,232,525,224,573,240,597,184,557,248,605
168,589,200,517,152,629,136,549,176,637,144,621,208,581,216,613
5,439,37,407,69,447,103,415,13,471,45,503,29,479,77,511
53,399,85,431,21,463,61,391,109,423,93,455,117,495,125,487
2,530,34,562,66,538,98,522,10,570,42,554,26,594,50,602
18,514,74,586,58,626,82,546,106,634,90,578,114,618,122,610
135,315,167,339,199,347,259,307,231,371,239,379,191,331,247,299
175,363,159,323,143,355,255,291,207,275,183,283,151,267,215,223
133,406,197,438,165,470,229,414,245,446,141,478,237,502,253,398
157,510,189,462,173,430,205,390,149,422,213,454,181,494,221,486
130,316,260,340,194,372,162,348,226,308,234,380,242,332,250,300
202,364,186,324,154,356,138,292,170,276,178,284,210,218,268,146
4,535,36,543,68,567,100,527,12,599,44,575,28,559,76,607
52,591,20,631,60,639,84,519,108,623,92,551,116,583,124,615
1,403,65,435,33,411,97,443,9,467,25,499,105,507,41,475
73,395,81,427,57,459,17,419,113,491,49,387,89,451,121,483
6,436,102,468,70,404,38,412,14,444,46,476,110,508,78,500
86,396,30,428,62,460,54,492,118,420,22,452,94,388,126,484
129,563,193,531,161,571,225,539,241,595,233,523,249,603,185,555
153,587,169,627,137,635,201,619,177,515,145,579,209,547,217,611
7,405,71,469,39,437,103,413,47,445,15,509,79,477,31,501
111,397,63,461,55,429,87,421,23,493,119,389,95,453,127,485
134,402,198,434,166,410,230,442,238,466,174,506,158,394,246,474
190,498,254,426,142,458,150,386,182,418,206,490,214,450,222,482
128,533,192,541,160,565,232,525,224,573,240,597,184,557,248,605
168,589,200,517,152,629,136,549,176,637,144,621,208,581,216,613
3,440,35,432,67,400,99,408,11,464,43,496,27,472,51,504
19,392,75,424,59,456,83,384,107,416,91,488,115,448,123,480
132,401,196,441,164,409,228,433,236,465,204,473,244,393,188,497
252,505,140,425,212,457,156,385,172,417,180,449,148,489,220,481
131,534,195,542,163,566,227,526,235,574,203,598,243,558,187,606
251,590,211,630,179,638,139,622,155,550,171,518,219,582,147,614
761,660,737,652,705,668,745,692,673,700,641,684,713,644,753,724
729,732,681,756,721,716,764,676,697,748,689,657,740,665,649,708
760,528,736,536,704,560,744,520,672,568,712,592,752,552,640,600
728,584,680,624,720,512,696,632,688,616,664,544,608,656,648,576
762,659,738,651,706,667,746,643,714,691,674,699,754,683,730,723
722,731,763,658,642,755,739,675,707,650,682,715,698,666,690,747
257,345,265,313,281,305,273,337,609,369,577,377,617,329,513,529
545,297,633,361,625,321,585,537,601,289,553,353,593,521,569,561
256,373,261,341,264,349,280,317,272,381,269,309,285,333,277,365
352,301,320,325,288,357,328,304,360,312,376,293,296,368,336,344
258,338,266,346,282,314,274,370,766,306,710,378,742,330,678,362
646,298,750,322,718,354,758,290,734,662,686,670,726,702,694,654
262,375,263,343,270,311,271,351,286,319,278,342,287,350,279,374
294,318,358,383,359,310,295,382,326,303,327,367,366,335,302,334
765,661,709,663,741,653,711,669,767,655,743,671,749,695,679,703
677,727,751,693,647,701,717,687,757,685,733,725,719,735,645,759
```

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

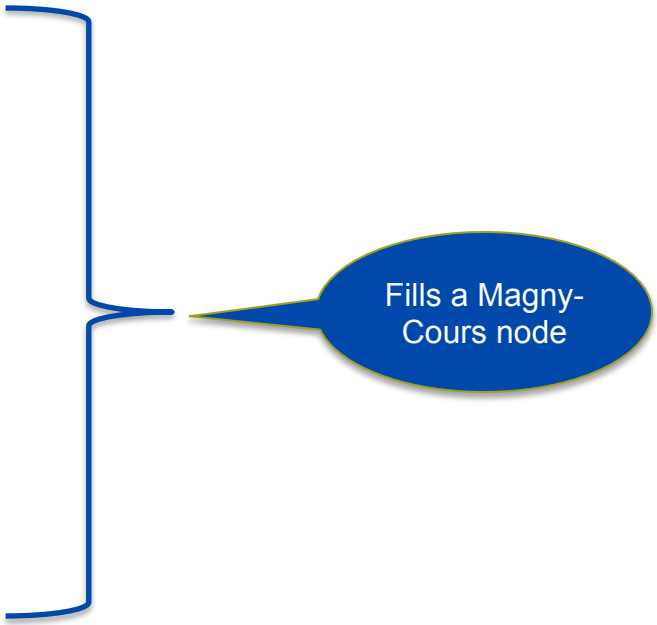
```
0,2,4,6,8,10,12,14,16,18,20,22,24,26,28,30
1,3,5,7,9,11,13,15,17,19,21,23,25,27,29,31
```

- 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
```



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

- **Default thread ordering**
  - Application 8538980 resources: utime ~126s, stime ~108s
- **Maximized on-node data movement with reordering**
  - Application 8538982 resources: utime ~38s, stime ~106s

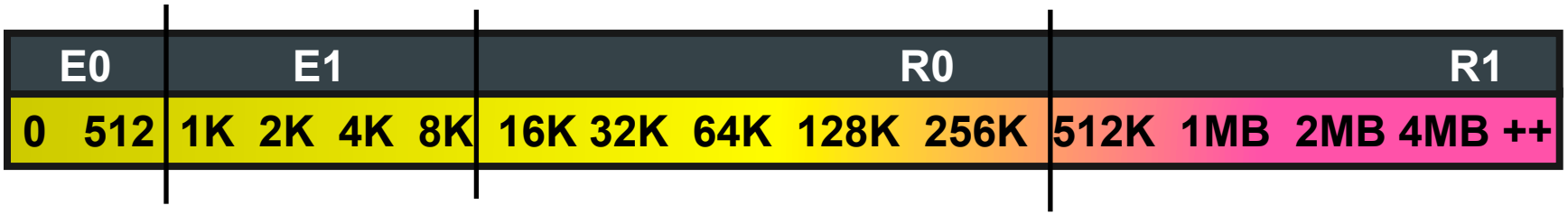
Some Useful Environment Variables

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

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

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
- 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*.

MPI-3 Support

- **MPI-3 Forum active participant**
  - Contributing to Fortran 2008 part of MPI-3
- **Tracking ANL MPICH2 implementation of MPI-3**
- **Cray plans to support MPI-3 incrementally (some functionality is available now)**
- **Will provide optimizations focused to certain areas such as with non-blocking collectives**

MPI-3 Features

- **Non-blocking collectives (MPT 5.6.0)**
  - 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-3 Features (2)

- Features planned for availability in 2013/2014

- Tools interface
- Mprobe
- MPI-3 const bindings
- Neighborhood collectives (functional)
- RMA (functional)
- Consistent use of []
- MPI_Count
- Fortran 2008 support
  - Needs compiler support

- Neighborhood collectives (optimized for topology)
- RMA (optimized to use DMAPP)

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

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

Questions ?