Optimising Communication on the Cray XE6

Outline

MPICH2 Releases for XE

Day in the Life of an MPI Message

- Gemini NIC Resources
- Eager Message Protocol
- Rendezvous Message Protocol
- Important MPI environment variables

MPI rank order

Huge Pages

Gemini Software

MPI device for Gemini based on

- User level Gemini Network Interface (uGNI)
- Distributed Memory Applications (DMAPP) library

FMA (Fast Memory Access)

- In general used for small transfers
- FMA transfers are lower latency

BTE (Block Transfer Engine)

- BTE transfers take longer to start but can transfer large amount of data without CPU involvement (DMA offload engine)

Which is Better

FMA PROS

- Lowest latency (<1.4 usec)
- All data has been read by the time dmapp returns
- More than one transfer active at the same time

FMA CONS

- CPU involved in the transfer
- Performance can vary depending on die used

BTE PROS

- Transfer done by Gemini, asynchronous with CPU
  - Transfers are queued if Gemini is busy
- Seems to get better P2P bandwidth in more cases

BTE CONS

- Higher latency : ~2 usec if queue is empty
  - Transfers are queued if Gemini is busy
- Only 4 BTE transaction can be active at a time
  (4 virtual channels)

MPICH2 and Cray MPT

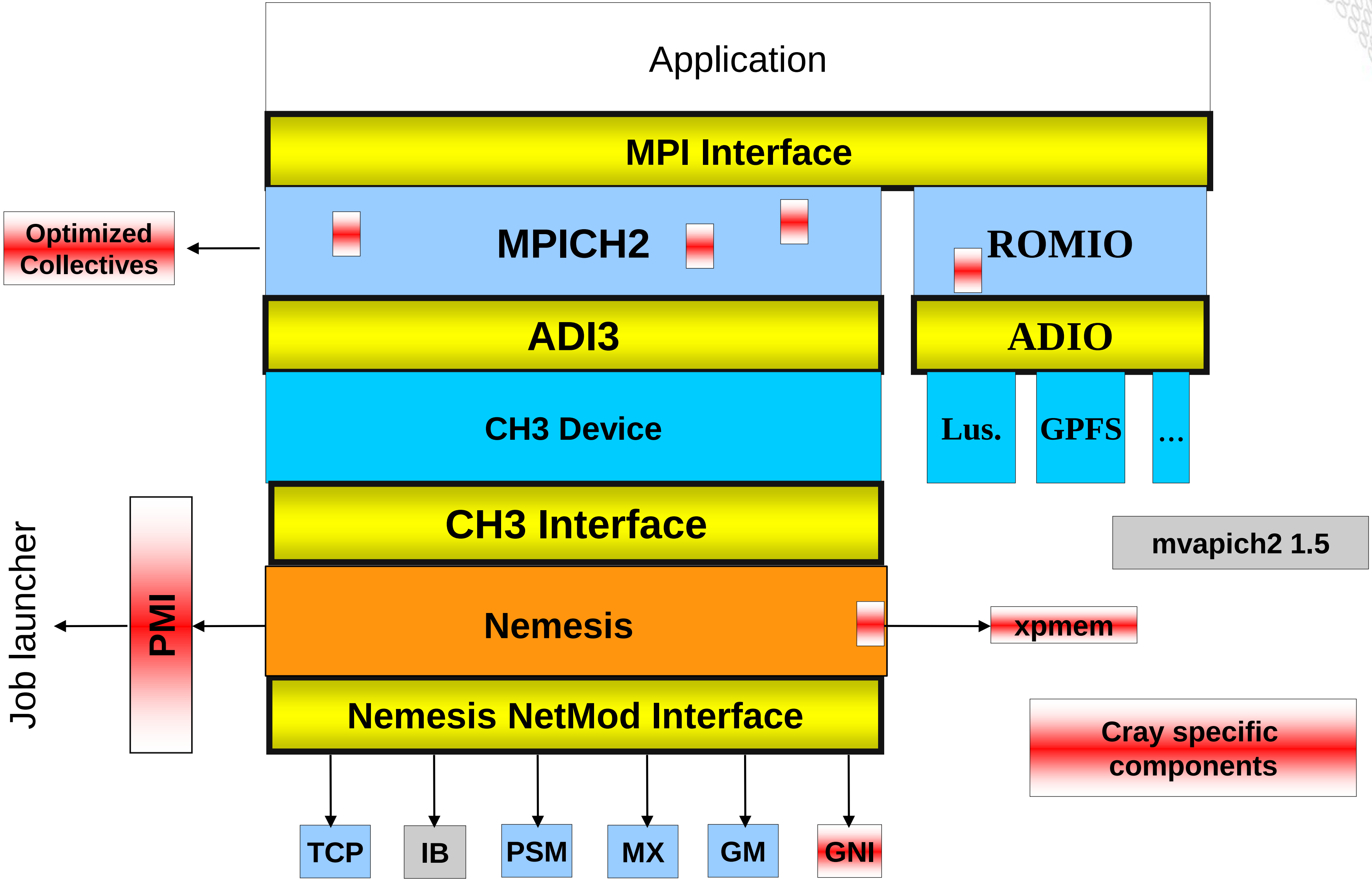
Cray MPI uses MPICH2 distribution from Argonne

- Provides a good, robust and feature rich MPI
- Cray provides low level communication libraries
- Point to point tuning
- Collective tuning
- Shared memory device is built on top of Cray XPMEM

Many layers are straight from MPICH2

- Error messages can be from MPICH2

MPICH2 on CRAY XE6

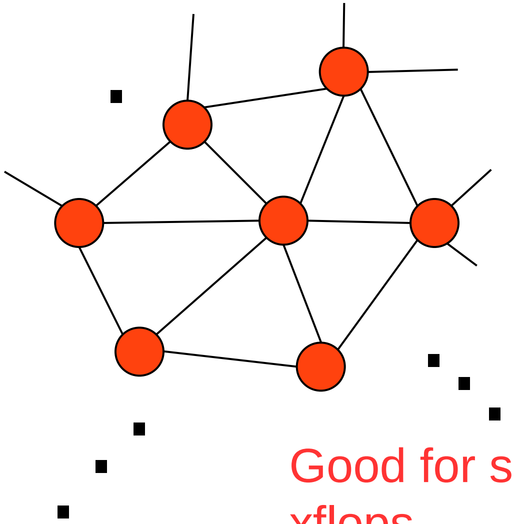


MPICH2 over GNI Basics

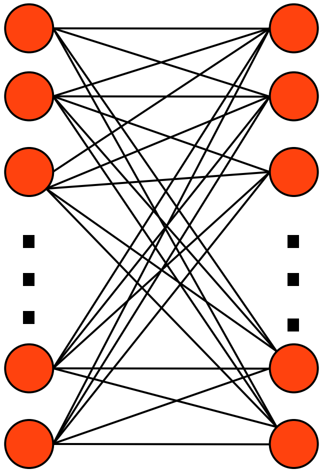
Implications of GNI SMSG Mailboxes for Apps

Applications with scalable communication patterns benefit from high message rates and low latency of GNI *private* SMSG mailboxes without large memory consumption. (~1.7 μsec latency nn, ~1.4 MM/sec nn/rank*)

Applications with dense communication graphs aren't going to scale too well on Gemini using GNI *private* SMSG mailboxes, may be okay with *shared* SMSG mailboxes.



Good for scaling to xflops



* For ranks on die0

okay for ISV apps scaling to ~1000 ranks, bad for petascale apps, forget xflops with MPI flat program model

A Day in the Life of an MPI Message

Gemini NIC Resources for Transferring Data

Eager Message Protocol : E0 and E1 Paths

Rendezvous Message Protocol : R0 and R1 Paths

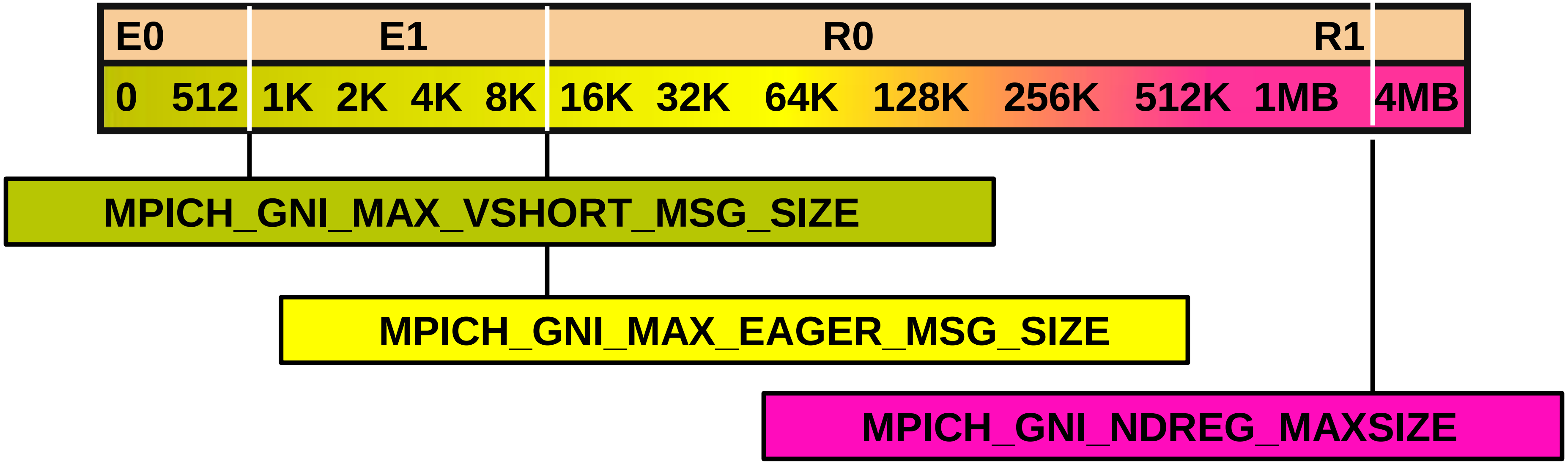
MPI environment variables that alter those paths

Day in the Life of an MPI Message

Four Main Pathways through the MPICH2 GNI NetMod

- Two EAGER paths (E0 and E1)
  - For a message that can fit in a GNI SMSG mailbox (E0)
  - For a message that can't fit into a mailbox but is less than MPICH_GNI_MAX_EAGER_MSG_SIZE in length (E1)
- Two RENDEZVOUS (aka LMT) paths : R0 (RDMA get) and R1 (RDMA put)

Selected Pathway is based 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: 216-984 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 4MB 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

- This 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
  - 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

EAGER Message Protocol

Default mailbox size varies with number of ranks in the job

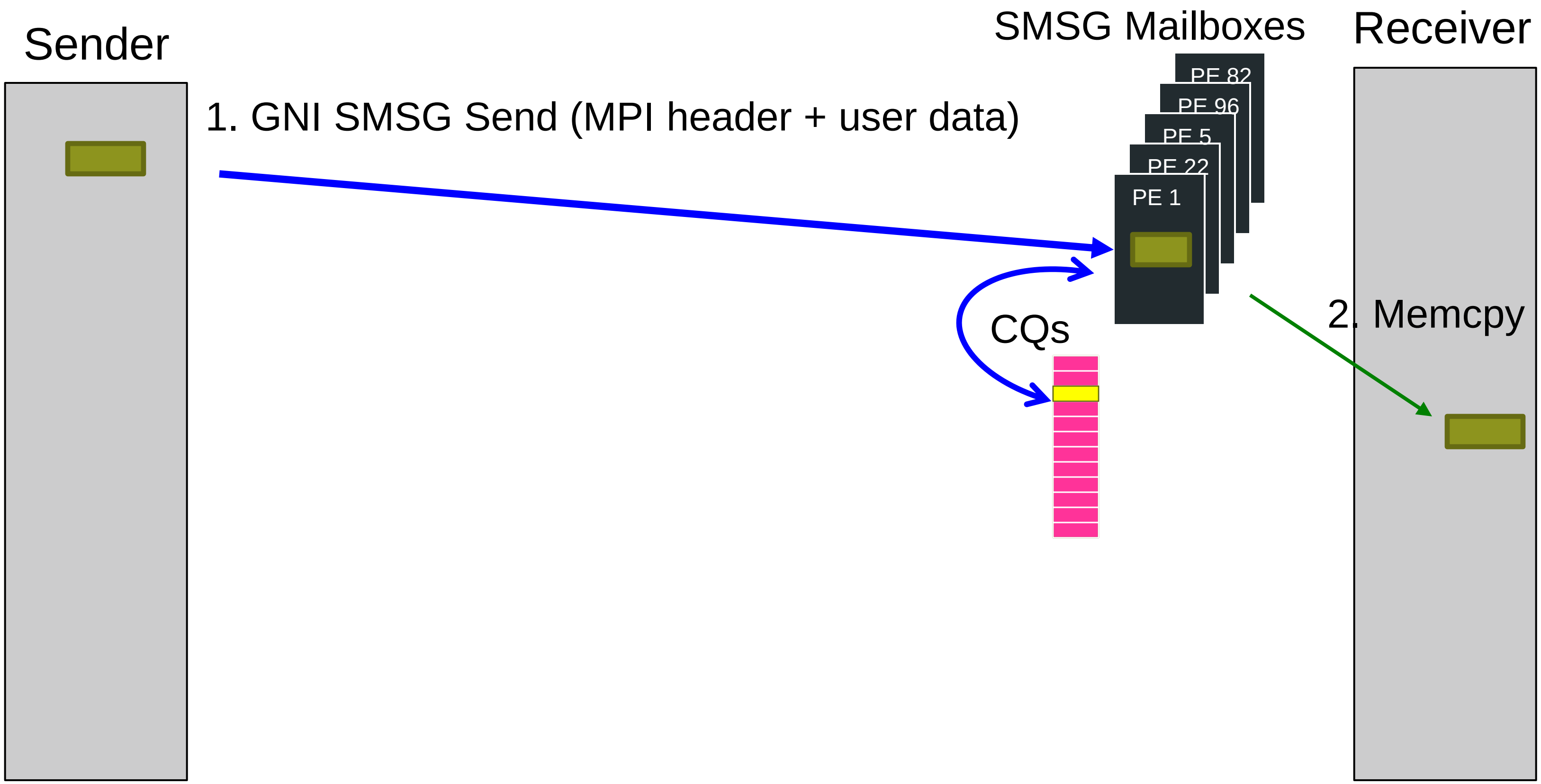
Protocol for messages that can fit into a GNI SMSG mailbox

The default varies with job size, although this can be tuned by the user to some extent

Ranks in Job	Max user data (MPT 5.3)	MPT 5.4 and later
< = 512 ranks	984 bytes	8152 bytes
> 512 and <= 1024	984 bytes	2008 bytes
> 1024 and < 16384	472 bytes	472 bytes
> 16384 ranks	216 bytes	216 bytes

Day in the Life of Message type E0

EAGER messages that fit in the GNI SMSG Mailbox

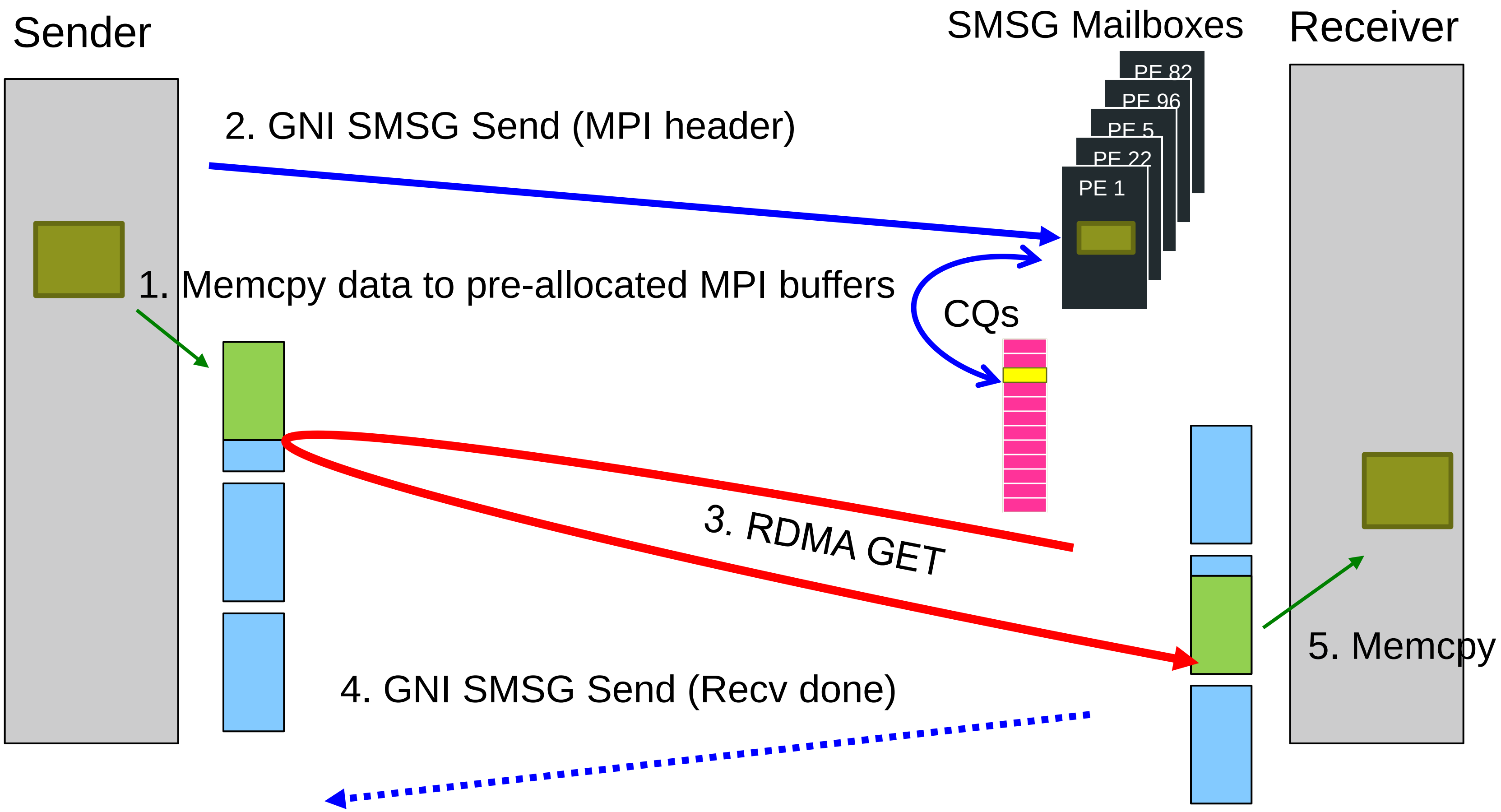


GNI SMSG Mailbox size changes with number of ranks in job

If user data is 16 bytes or less, it is copied into the MPI header

Day in the Life of 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
5. Memcpy
4. GNI SMSG Send (Recv done)

User data is copied into internal MPI buffers on both send and receive side

MPICH_GNI_NUM_BUFS
default 64 buffers, each 32K

RENDEZVOUS Message Protocol

Data is transferred after both MPI_Send and MPI_Recv are encountered

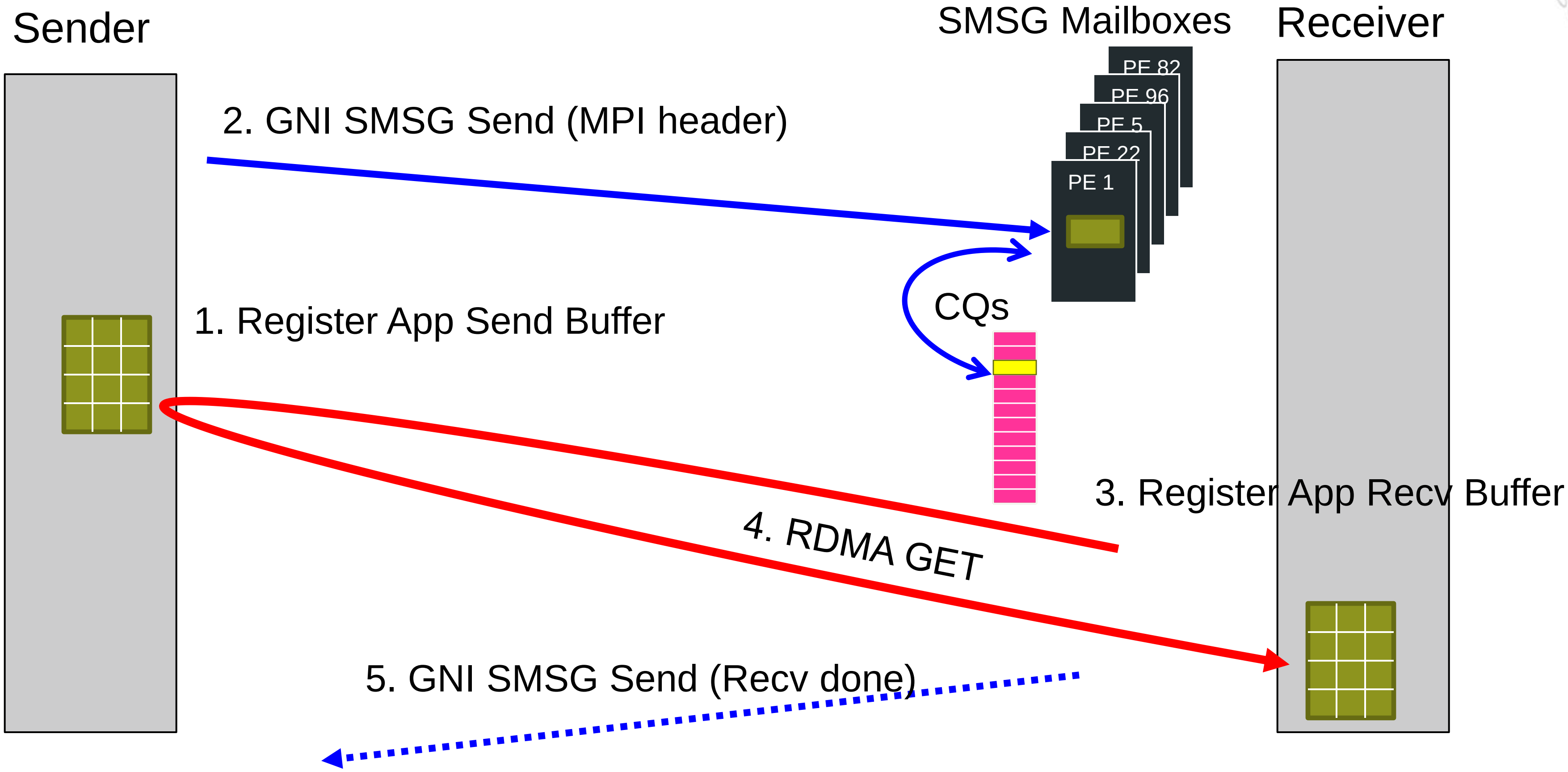
Two RENDEZVOUS Pathways

- R0 – **RDMA GET** method
  - By default, used for messages between 8K and 4MB
  - 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 and computation in this path
    - Helps to issue MPI_Isend **prior** to MPI_Irecv
  - May deplete memory registration resources

- R1 – Pipelined **RDMA PUT** method
  - By default, used for messages greater than 512k
  - Use **MPICH_GNI_NDREG_MAXSIZE** to adjust starting point
  - Very difficult to overlap communication and computation in this path

Day in the Life of Message type R0

Rendezvous messages using RDMA Get

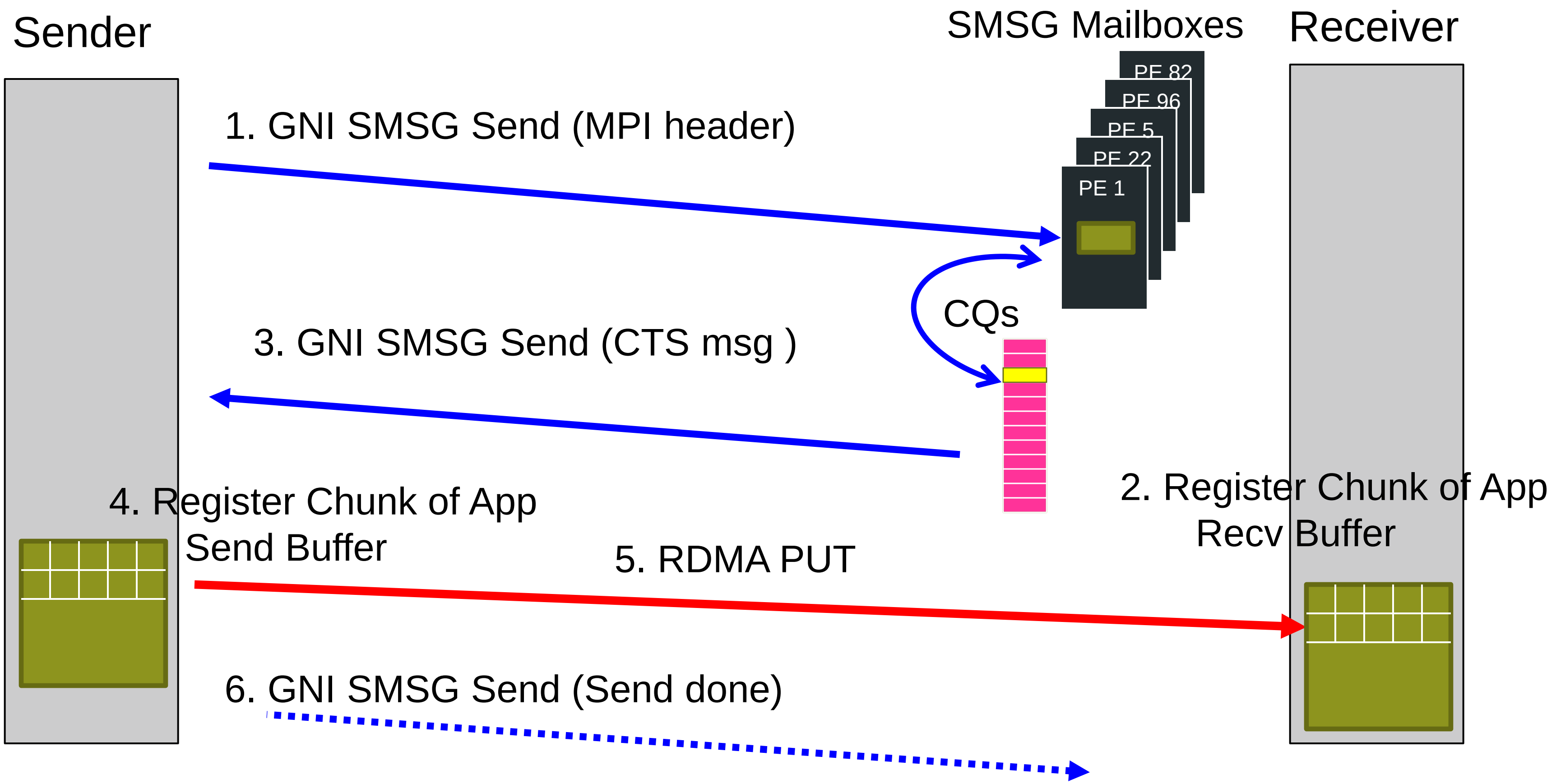

Sender
SMSG Mailboxes
Receiver
PE 82
PE 96
PE 5
PE 22
PE 1
2. GNI SMSG Send (MPI header)
CQs
1. Register App Send Buffer
3. Register App Recv Buffer
4. RDMA GET
5. GNI SMSG Send (Recv done)

No extra data copies

Best chance of overlapping communication with computation

Day in the Life of 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)

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 4MB)

Environment Variables

Environment Variables for Inter-node Point-to-Point Messaging

Check ‘man mpi’ for all details

MPICH_GNI_MAX_VSHORT_MSG_SIZE

Can be used to control the maximum size message that can go through the private SMSG mailbox protocol (E0 *eager* path).

Default varies with job size

Maximum size is 8192 bytes. Minimum is 80 bytes.

If you are trying to demonstrate an MPI_Alltoall at very high count, with smallest possible memory usage, may be good to set this as low as possible.

If you know your app has a scalable communication pattern, and the performance drops at one of the edges shown on table on slide 18, you may want to set this environment variable.

Pre-posting receives for this protocol avoids a potential extra memcpy at the receiver.

MPICH_GNI_MAX_EAGER_MSG_SIZE

Default is 8192 bytes

Maximum size message that go through the *eager* (E1) 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.

Maximum allowable setting is 131072 bytes

Pre-posting receives can avoid potential double memcpy at the receiver.

Note that a 40-byte Nemesis header is included in account for the message size.

MPICH_GNI_MBOX_PLACEMENT

Provides a means for controlling which memories on a node are used for some SMSG mailboxes (private).

Default is to place the mailboxes on the memory where the process is running when the memory for the mailboxes is faulted in.

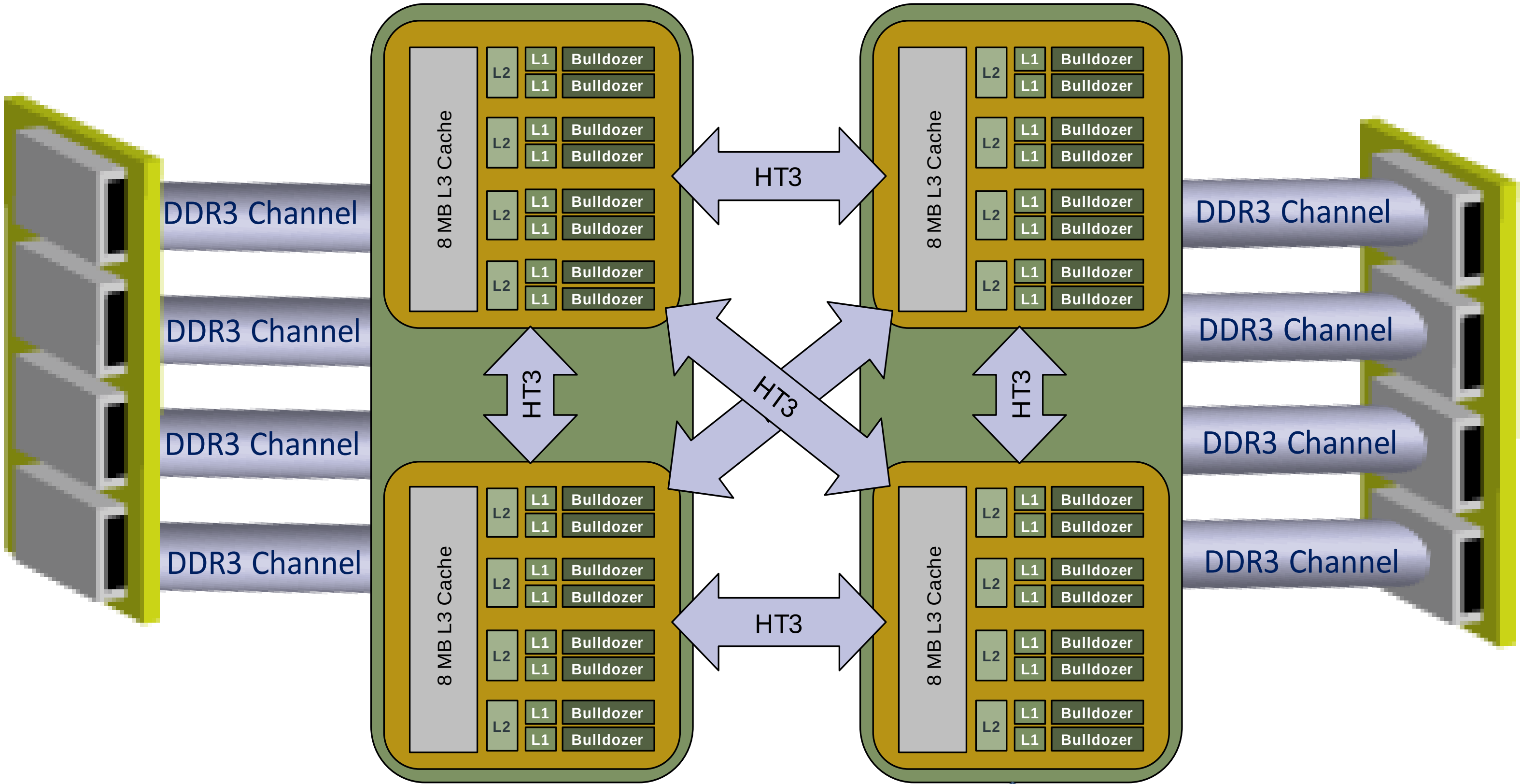
For optimal MPI message rates, better to place mailboxes on memory of die0 (where Gemini is attached).

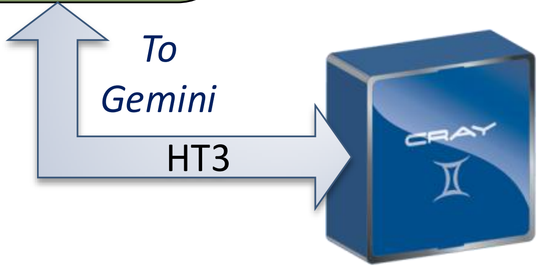
Only applies to first 4096 mailboxes of each rank on the node.

Syntax for enabling placement of mailboxes near the Gemini:

```
export MPICH_GNI_MBOX_PLACEMENT=nic
```

Cray XE6 Node Details – 32-core Interlagos





- 2 Multi-Chip Modules,
  4 Opteron Dies: ~300 Gflops
- 8 Channels of DDR3 Bandwidth to 8 DIMMs: ~105 GB/s
- 32 Computational Cores, 32 MB of L3 cache
- Dies are fully connected with HT3

MPICH_GNI_RDMA_THRESHOLD

Default is now 1024 bytes

Controls the threshold at which the GNI netmod switches from using FMA for RDMA read/write operations to using the BTE.

Since BTE is managed in the kernel, BTE initiated RDMA requests can progress even if the applications isn’t in MPI.

Owing to Opteron/HT quirks, the BTE is often better for moving data to/from memories that are farther from the Gemini.

But using the BTE may lead to more interrupts being generated

MPICH_GNI_NDREG_LAZYMEM

Default is enabled. To disable

```
export MPICH_GNI_NDREG_LAZYMEM=disabled
```

Controls whether or not to use a lazy memory deregistration policy inside UDREG. Memory registration is expensive so this is usually a good idea.

Only important for those applications using the LMT (large message transfer) path, i.e. messages greater than `MPICH_GNI_MAX_EAGER_MSG_SIZE.`

Disabling results in a significant drop in measured bandwidth for large transfers ~40-50 %.

If code only works with this feature being diabled => BUG

MPICH_GNI_DMAPP_INTEROP

Only relevant for mixed MPI/SHMEM/UPC/CAF codes

Normally want to leave enabled so MPICH2 and DMAPP can share the same memory registration cache,

May have to disable for SHMEM codes that call *shmem_init* after *MPI_Init.*

May want to disable if trying to add SHMEM/CAF to an MPI code and notice a big performance drop.

Syntax:

```
export MPICH_GNI_DMAPP_INTEROP=disabled
```

MPICH_GNI_DMAPP_INTEROP

May have to set to disable if one gets a traceback like this:

```
Rank 834 Fatal error in MPI_Alltoall: Other MPI error, error stack:
MPI_Alltoall(768)......................: MPI_Alltoall(sbuf=0x2aab9c301010,
scount=2596, MPI_DOUBLE, rbuf=0x2aab7ae01010, rcount=2596,
MPI_DOUBLE,
comm=0x84000004) failed
MPIR_Alltoall(469).....................:
MPIC_Isend(453).........................:
MPID_nem_lmt_RndvSend(102).............:
MPID_nem_gni_lmt_initiate_lmt(580)......: failure occurred while attempting to
send RTS packet
MPID_nem_gni_iStartContigMsg(869).......:
MPID_nem_gni_iSendContig_start(763).....:
MPID_nem_gni_send_conn_req(626).........:
MPID_nem_gni_progress_send_conn_req(193):
MPID_nem_gni_smsg_mbox_alloc(357).......:
MPID_nem_gni_smsg_mbox_block_alloc(268).: GNI_MemRegister
GNI_RC_ERROR_RESOURCE)
```

MPICH_GNI_NUM_BUFS

Default is 64 32K buffers (2M total)

Controls the number of 32KB DMA buffers available for each rank to use in the GET-based Eager protocol (E1).

May help to modestly increase. But other resources constrain the usability of a large number of buffers, so don’t go berserk with this one.

Syntax:

```
export MPICH_GNI_NUM_BUFS=X
```

MPICH_GNI_DYNAMIC_CONN

Enabled by default

Normally want to leave enabled so mailbox resources (memory, NIC resources) are allocated only when the application needs them

If application does all-to-all or many-to-one/few, may as well disable dynamic connections. This will result in significant startup/shutdown costs though.

Syntax for disabling:

```
export MPICH_GNI_DYNAMIC_CONN=disabled
```

Environment Variables for Collective Operations

Again, please check ‘man mpi’

MPI_Allgather

With MPT 5.1 switched to using Seastar-style algorithm where for short transfers/rank: use MPI_Gather/MPI_Bcast rather than ANL algorithm

Switchover from Cray algorithm to ANL algorithm can be controlled by the `MPICH_ALLGATHER_VSHORT_MSG` environment variable. By default enabled for transfers/rank of 1024 bytes or less

The Cray algorithm can be deactivated by setting

```
export MPICH_COLL_OPT_OFF=mpi_allgather
```

MPI_Allgatherv

With MPT 5.1 switched to using Seastar-style algorithm where for short transfers/rank: use a specialized MPI_Gatherv/MPI_Bcast rather than ANL algorithm

Switchover from Cray algorithm to ANL algorithm can be controlled by the `MPICH_ALLGATHERV_VSHORT_MSG` environment variable. By default enabled for transfers/rank of 1024 bytes or less.

The Cray algorithm can be deactivated by setting

```
export MPICH_COLL_OPT_OFF=mpi_allgatherv
```

MPI_Alltoall

Optimizations added in MPT 5.1

Switchover from ANL's implementation of Bruck algorithm (IEEE TPDS, Nov. 1997) is controllable via the `MPICH_ALLTOALL_SHORT_MSG` environment variable. Defaults are

ranks in communicator	Limit (in bytes) for using Bruck
<= 512	2048
>512 && <=1024	1024
> 1024	128

- Larger transfers use an optimized pair-wise exchange algorithm
- New algorithm can be disabled by
  `export MPICH_COLL_OPT_OFF=mpi_alltoall`

MPI_Allreduce/MPI_Reduce

The ANL smp-aware MPI_Allreduce/MPI_Reduce algorithms can cause issues with bitwise reproducibility. To address this Cray MPICH2 has two new environment variables starting with MPT 5.1 -

`MPI_ALLREDUCE_NO_SMP` – disables use of smp-aware MPI_Allreduce

`MPI_REDUCE_NO_SMP` – disables use of smp-aware MPI_Reduce

MPI_Bcast

Starting with MPT 5.1, all ANL algorithms except for binomial tree are disabled since the others perform poorly for communicators with 512 or more ranks

To disable this tree algorithm-only behaviour, set the MPICH_BCAST_ONLY_TREE environment variable to 0, i.e.

```
export MPICH_BCAST_ONLY_TREE=0
```

Environment Variables for Intra-node Point-to-Point Messaging

Did I mention ‚man mpi‘ ?

MPICH_SMP_SINGLE_COPY_SIZE

Default is 8192 bytes

Specifies threshold at which the Nemesis shared memory channel switches to a single-copy, XPMEM based protocol for intra-node messages

Note that XPMEM is completely different from KNEM (INRIA) which is in use in MPICH2 deployed on other Linux cluster systems. Don’t get confused if following mpich-discuss, etc.

MPICH_SMP_SINGLE_COPY_OFF

In MPT 5.1 the default is enabled, In 5.4 it is disabled

Specifies whether or not to use a XPMEM-based single-copy protocol for intra-node messaes of size MPICH_SMP_SINGLE_COPY_SIZE bytes or larger

May need to set this environment variable if

- Finding XPMEM is kernel OOPses (check the console on the SMW)
- Sometimes helps if hitting UDREG problems. XPMEM goes kind of crazy with Linux mmu notifiers and causes lots of UDREG invalidations (at least the way MPICH2 uses XPMEM).

MPICH_NEMESIS_ASYNC_PROGRESS

The Gemini does not have a progress engine.

If set, enables the MPICH2 asynchronous progress feature. In addition, the MPICH_MAX_THREAD_SAFETY environment variable must be set to multiple in order to enable this feature. The asynchronous progress feature is effective only when the aprun -r option is used to reserve one or more processors on each node for core specialization. See the aprun(1) man page for more information regarding how to use core specialization.

MPI rank orders

Is your nearest neighbor really your nearest neighbor? And do you want them to be your nearest neighbor?

Rank Placement

The default ordering can be changed using the following environment variable:

- MPICH_RANK_REORDER_METHOD

These are the different values that you can set it to:

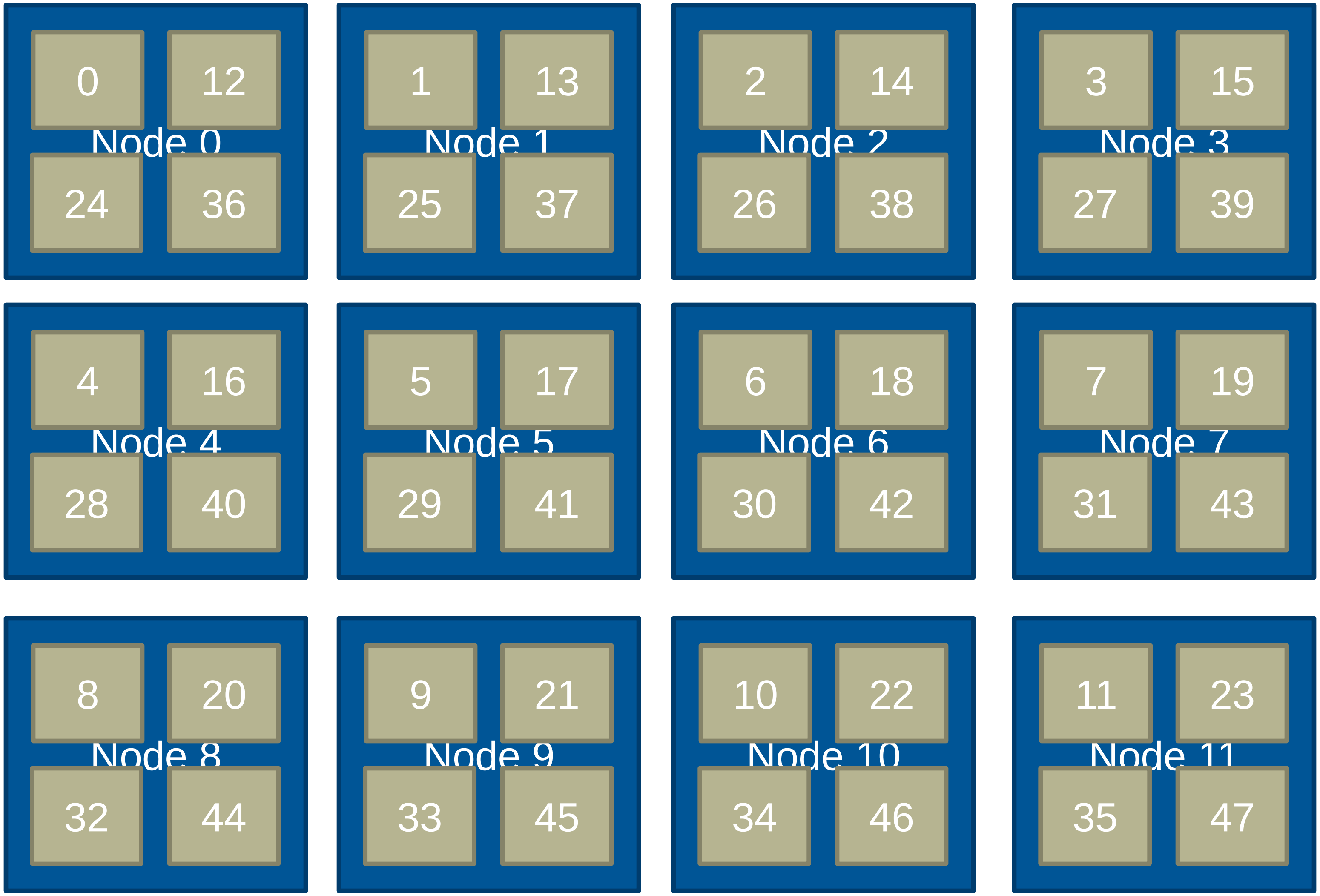
- 0: Round-robin placement – Sequential ranks are placed on the next node in the list. Placement starts over with the first node upon reaching the end of the list.
- 1: (DEFAULT) SMP-style placement – Sequential ranks fill up each node before moving to the next.
- 2: Folded rank placement – Similar to 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.

Rank Placement

When is this useful?

- Point-to-point communication consumes a significant fraction of program time and a load imbalance detected
- Also shown to help for collectives (alltoall) on subcommunicators
- Spread out IO across nodes

0: Round Robin Placement



Node 0: 0 12 24 36

Node 1: 1 13 25 37

Node 2: 2 14 26 38

Node 3: 3 15 27 39

Node 4: 4 16 28 40

Node 5: 5 17 29 41

Node 6: 6 18 30 42

Node 7: 7 19 31 43

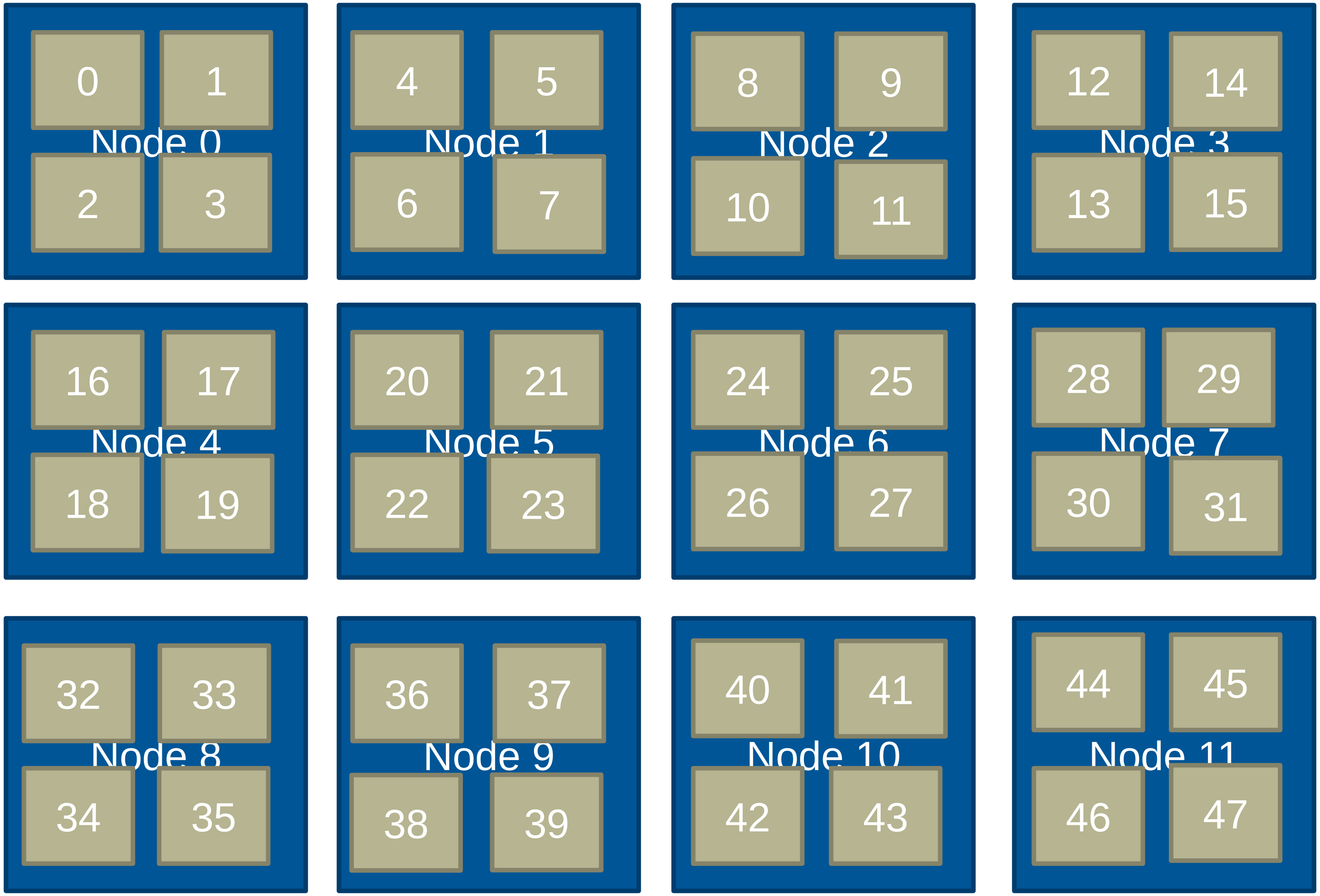
Node 8: 8 20 32 44

Node 9: 9 21 33 45

Node 10: 10 22 34 46

Node 11: 11 23 35 47

1: SMP Placement (default)



0 1
Node 0
2 3

4 5
Node 1
6 7

8 9
Node 2
10 11

12 14
Node 3
13 15

16 17
Node 4
18 19

20 21
Node 5
22 23

24 25
Node 6
26 27

28 29
Node 7
30 31

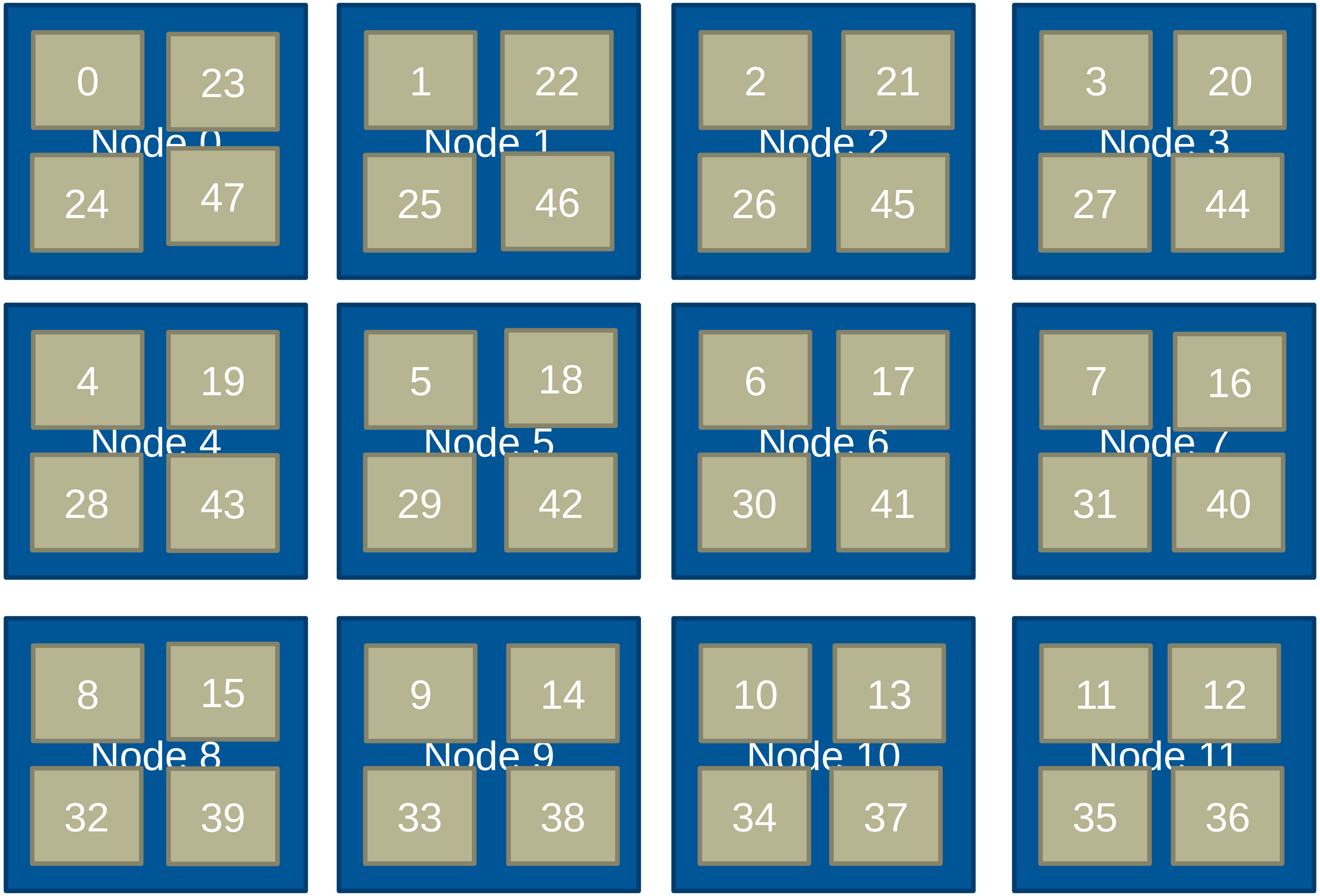
32 33
Node 8
34 35

36 37
Node 9
38 39

40 41
Node 10
42 43

44 45
Node 11
46 47

2: Folded Placement



0 23
Node 0
24 47

1 22
Node 1
25 46

2 21
Node 2
26 45

3 20
Node 3
27 44

4 19
Node 4
28 43

5 18
Node 5
29 42

6 17
Node 6
30 41

7 16
Node 7
31 40

8 15
Node 8
32 39

9 14
Node 9
33 38

10 13
Node 10
34 37

11 12
Node 11
35 36

Rank Placement

From the man page: The grid_order utility is used to generate a rank order list for use by an MPI application that uses communication between nearest neighbors in a grid. When executed with the desired arguments, grid_order generates rank order information in the appropriate format and writes it to stdout. This output can then be copied or written into a file named MPICH_RANK_ORDER and used with the

MPICH_RANK_REORDER_METHOD=3

environment variable to override the default MPI rank placement scheme and specify a custom rank placement.

Craypat will also make suggestions

Case Study:
AWP-ODC and MPI Re-ordering

David Whitaker
Applications Group
Cray Inc

AWP-ODC and MPI re-ordering

AWP-ODC code from NCAR procurement

- Earthquake code – x, y, z structured grid

MPI uses two communicators

- 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

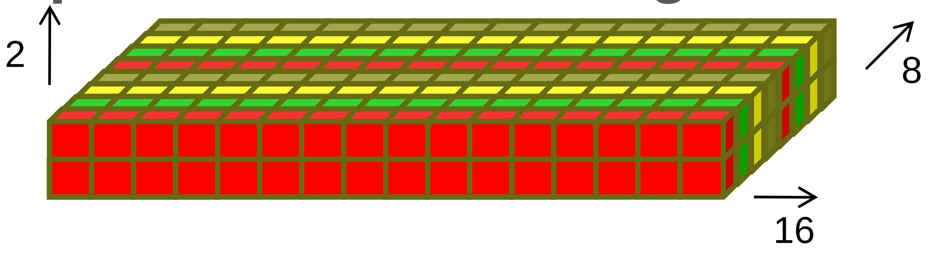
AWP-ODC example – Part 1

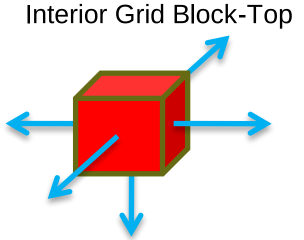
NCAR provided three test cases:

- 256 processors: 16x2x8 grid
- 512 processors: 16x4x8 grid
- 1024 processors: 16x4x16 grid

Graphics by Kevin McMahon

For 256 processors: 16x2x8 grid





- 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
```

IMD

```
+ grid_order -R -P -m 8192 -n 32 -g 32,16,16 -c 4,4,2
outnbl16_8192.N32:4.031155e-06 cpuseconds per step and atom
outnbl64_8192.N32:4.095462e-06 cpuseconds per step and atom

+ grid_order -R -P -m 8192 -n 32 -g 32,16,16 -c 2,4,1
outnbl16_8192.N32:4.026443e-06 cpuseconds per step and atom
outnbl64_8192.N32:4.039731e-06 cpuseconds per step and atom

+ grid_order -R -P -m 8192 -n 32 -g 32,16,16 -c 4,4,1
outnbl16_8192.N32:3.902369e-06 cpuseconds per step and atom
outnbl64_8192.N32:4.027267e-06 cpuseconds per step and atom
```

Hugepages

Why use Huge Pages

The Gemini perform better with HUGE pages than with 4K pages.

HUGE pages use less GEMINI resources than 4k pages (fewer bytes).

Your code may run with fewer TLB misses (hence faster)

Use modules to change default page sizes (man intro_hugepages):

- e.g. module load craype-hugepages#
  - craype-hugepages128K
  - craype-hugepages512K
  - craype-hugepages2M
  - craype-hugepages8M
  - craype-hugepages16M
  - craype-hugepages64M

What is in the module file

The setting of HUGETLB_DEFAULT_PAGE_SIZE varies between hugepages modules, but the other settings are the same

```
hpcander@eslogin003:~> module show craype-hugepages8M
-------------------------------------------------------------------
/opt/cray/xt-asyncpe/default/modulefiles/craype-hugepages8M:
append-path      PE_PRODUCT_LIST HUGETLB8M
setenv           HUGETLB_DEFAULT_PAGE_SIZE 8M
setenv           HUGETLB_MORECORE yes
setenv           HUGETLB_ELFMAP W
setenv           HUGETLB_FORCE_ELFMAP yes+
setenv           HUGETLB8M_POST_LINK_OPTS -Wl,--whole-archive,-lhugetlbfs,--
no-whole-archive -Wl,-Ttext-segment=0x4000000,-zmax-page-size=0x4000000
-------------------------------------------------------------------
```

So to use huge pages, it is important to link your executable with a huge pages module loaded, but you can select a different one when running.

Huge pages – details

Linked with the correct library: `-lhugetlbfs`

Activate the library at run time: `export HUGETLB_MORECORE=yes`

Launch the program with aprun pre-allocating huge pages

- request <size> MBytes per PE `-m<size>h` (advisory mode)
- request <size> MBytes per PE `-m<size>hs` (required mode)
  - What if the request can't be satisfied ? Slow or crash ?

Example: `aprun -m700hs -N2 -n8 ./my_app`

- Requires 1400 MBytes of huge page memory on each node

Thank You
Questions ?