# MPI Optimization and Tips

# Outline



Go to Menu

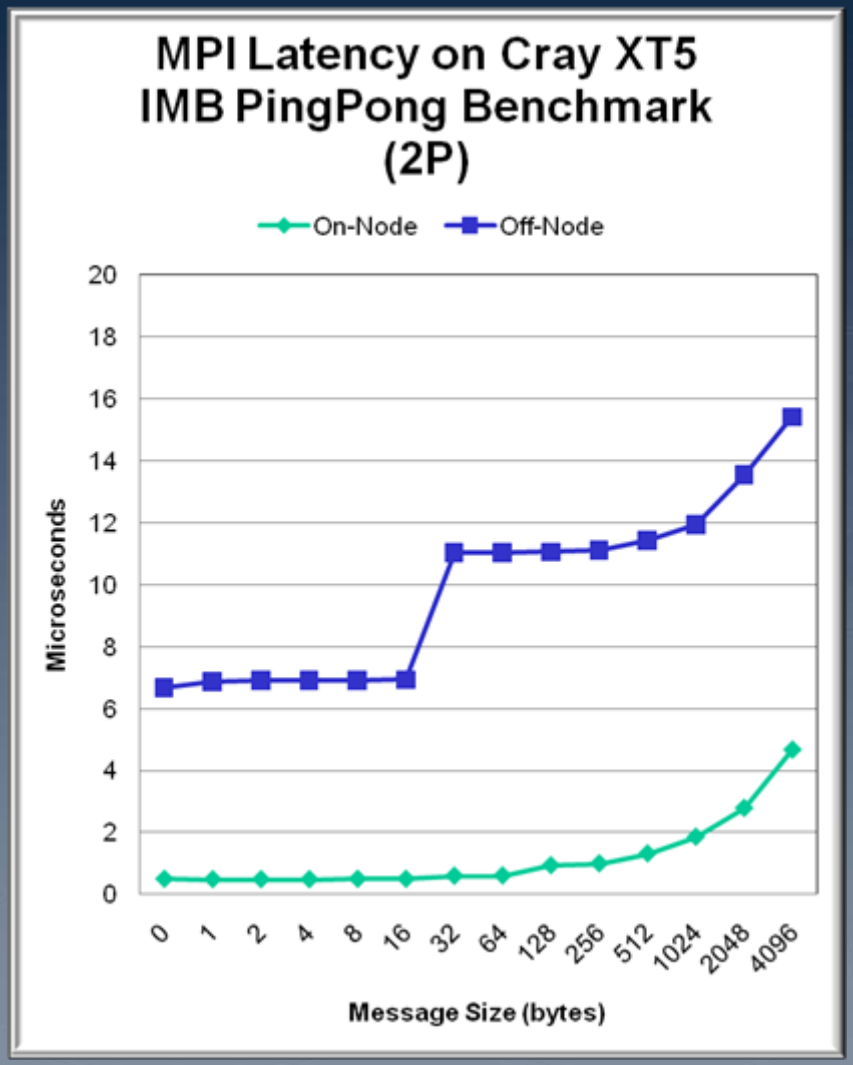
# Outline: Cray Message Passing Toolkit (MPT)

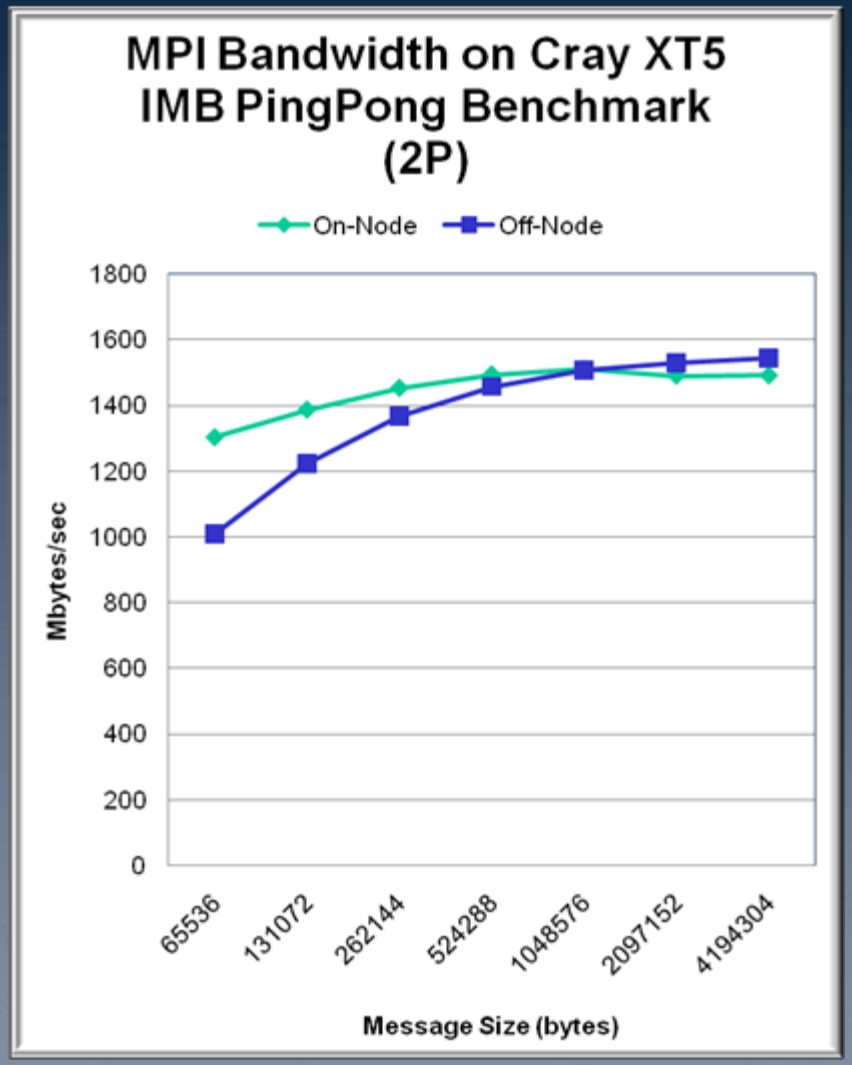
# Cray Message Passing Toolkit (MPT) 3.2.0

- Cray Message Passing Toolkit (MPT) - Cray's MPI library
- Toolkit includes MPI and SHMEM
  - MPI based on MPICH2 version 1.0.6 from ANL
  - Support for multiple compilers (CCE, PGI, Pathscale, GNU)
  - Numerous Cray enhancements and optimizations
- What Unique Features does CRAY MPI provide for XT5?
  - Custom Portals device driver
  - Custom Shared Memory (SMP) device driver
  - Multi-device implementation for a single job
    - Optimal messaging path is selected automatically
  - Optimized Collectives
  - MPI I/O Enhancements
  - Support for up to 256,000 MPI ranks
  - Custom Process Manager Interface (PMI) for launching
    - Interfaces with existing ALPs software (aprun)
    - A PMI daemon process is started on each node
    - Support for Process-to-CPU affinity
    - Support for Rank Re-Ordering

Go to Menu

# IMB PingPong Benchmark



# Key Cray MPI Environment Variables

- Why use MPI environment variables?
  - Allow users to tweak optimizations for specific application behavior
  - Flexibility to choose cutoff values for collective optimizations
  - Determine maximum size of internal MPI resources - buffers/queues, etc.

- MPI Display Variables
  - **`export MPICH_VERSION_DISPLAY=1`**
    - Displays version of Cray MPI being used
    - strings ./mpi.exe | grep VERSION

    MPI VERSION : CRAY MPICH2 XT version 3.1.2 (ANL base 1.0.6)
    BUILD INFO : Built Mon Feb 16 10:20:17 2009 (svn rev 7304)

  - **`export MPICH_ENV_DISPLAY=1`**
    - Displays all MPI environment variables and their current values
    - Helpful to determine what defaults are set to

# MPI Environment Variables: Default Values

The default values of MPI environment variables:

```
MPI VERSION : CRAY MPICH2 XT version 3.1.2-
pre (ANL base 1.0.6)
BUILD INFO  : Built Thu Feb 26  3:58:36 2009
(svn rev 7308)
PE 0: MPICH environment settings:
PE 0:   MPICH_ENV_DISPLAY          = 1
PE 0:   MPICH_VERSION_DISPLAY      = 1
PE 0:   MPICH_ABORT_ON_ERROR       = 0
PE 0:   MPICH_CPU_YIELD            = 0
PE 0:   MPICH_RANK_REORDER_METHOD  = 1
PE 0:   MPICH_RANK_REORDER_DISPLAY = 0
PE 0:   MPICH_MAX_THREAD_SAFETY    = single
PE 0:   MPICH_MSGS_PER_PROC        = 16384
PE 0: MPICH/SMP environment settings:
PE 0:   MPICH_SMP_OFF              = 0
PE 0:   MPICH_SMPDEV_BUFS_PER_PROC = 32
PE 0:   MPICH_SMP_SINGLE_COPY_SIZE = 131072
PE 0:   MPICH_SMP_SINGLE_COPY_OFF  = 0
PE 0: MPICH/PORTALS environment settings:
PE 0:   MPICH_MAX_SHORT_MSG_SIZE   = 128000
PE 0:   MPICH_UNEX_BUFFER_SIZE     = 62914560
PE 0:   MPICH_PTL_UNEX_EVENTS      = 20480
PE 0:   MPICH_PTL_OTHER_EVENTS     = 2048
PE 0:   MPICH_VSHORT_OFF           = 0
PE 0:   MPICH_MAX_VSHORT_MSG_SIZE  = 1024
PE 0:   MPICH_VSHORT_BUFFERS       = 32
PE 0:   MPICH_PTL_EAGER_LONG       = 0
PE 0:   MPICH_PTL_MATCH_OFF        = 0
PE 0:   MPICH_PTL_SEND_CREDITS     = 0
PE 0: MPICH/COLLECTIVE environment settings:
PE 0:   MPICH_FAST_MEMCPY          = 0
PE 0:   MPICH_COLL_OPT_OFF         = 0
PE 0:   MPICH_COLL_SYNC            = 0
PE 0:   MPICH_BCAST_ONLY_TREE      = 1
PE 0:   MPICH_ALLTOALL_SHORT_MSG   = 1024
PE 0:   MPICH_REDUCE_SHORT_MSG     = 65536
PE 0:   MPICH_REDUCE_LARGE_MSG     = 131072
PE 0:   MPICH_ALLREDUCE_LARGE_MSG  = 262144
PE 0:   MPICH_ALLGATHER_VSHORT_MSG = 2048
PE 0:   MPICH_ALLTOALLVW_FCSIZE    = 32
PE 0:   MPICH_ALLTOALLVW_SENDWIN   = 20
PE 0:   MPICH_ALLTOALLVW_RECVWIN   = 20
PE 0: MPICH/MPIIO environment settings:
PE 0:   MPICH_MPIIO_HINTS_DISPLAY  = 0
PE 0:   MPICH_MPIIO_CB_ALIGN       = 0
PE 0:   MPICH_MPIIO_HINTS          = NULL
```

# Portals API

- API designed to fit MPI message matching rules
- Emphasis on application bypass, off loading of message passing work from application process
- Emphasis on scalability
- Similar in concept to Quadrics t-ports

# Cray MPI XT Portals Communications

- Short-message - **Eager** Protocol
  - The sending rank "pushes" the message to the receiving rank
  - Used for messages **MPICH_MAX_SHORT_MSG_SIZE** bytes or less
  - Sender assumes that receiver can handle the message
    - Matching receive is posted - or -
    - Has available event queue entries **MPICH_PTL_UNEX_EVENTS** and buffer space **MPICH_UNEX_BUFFER_SIZE** to store the message
- Long-message - **Rendezvous** Protocol
  - Messages are "pulled" by the receiving rank
  - Used for messages greater than **MPICH_MAX_SHORT_MSG_SIZE** bytes
  - Sender sends MPI Header with information for the receiver to pull over the data
  - Data is sent only after matching receive is posted by receiving rank

# Auto-Scaling MPI Environment Variables

- Key MPI variables that ***change*** their default values dependent on job size (number of MPI processes)

| | |
|---|---|
| **MPICH_MAX_SHORT_MSG_SIZE** | **MPICH_PTL_UNEX_EVENTS** |
| **MPICH_UNEX_BUFFER_SIZE** | **MPICH_PTL_OTHER_EVENTS** |

  - New in MPT 3.1
  - Aids in scaling applications
  - “Default” values are based on total number of ranks in job
  - See MPI man page for specific formulas used

- The automatic values are not always the best
  - Adjusted defaults aren't perfect for all applications
  - Assumes a somewhat communication-balanced application
  - Users should always test their applications and try to find the best values.

# Cray MPI XT Portals Communications: Default Values

Default values for various Cray MPI job sizes:

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

# What does this mean?

- If you see this error message:

internal ABORT - process 0: Other MPI error, error stack:

MPIDI_PortalsU_Request_PUPE(317): exhausted unexpected receive queue buffering increase via env. var. MPICH_UNEX_BUFFER_SIZE

- It means:

  The application is sending too many short, unexpected messages to a particular receiver.

- Try doing this to work around the problem:

  Increase the amount of memory for MPI buffering using the MPICH_UNEX_BUFFER_SIZE variable(default is 60 MB) and/or decrease the short message threshold using the MPICH_MAX_SHORT_MSG_SIZE (default is 128000 bytes) variable. May want to set MPICH_DBMASK to 0x200 to get a traceback/coredump to learn where in application this problem is occurring.

# What does this mean? (continued)

- If you see this error message:

Assertion failed in file /notbackedup/users/rsrel/rs64.REL_1_4_06.060419.Wed/pe/computelibs/mpich2/src/mpid/portals32/src/portals_init.c at line 193: MPIDI_Portals_unex_block_size > MPIDI_Portals_short_size

- It means:

  The appearance of this assertion means that the size of the unexpected buffer space is too small to contain even 1 unexpected short message.

- Try doing this to work around the problem:

  User needs to check their MPICH environment settings to make sure there are no conflicts between the setting of the MPICH_UNEX_BUFFER_SIZE variable and the setting for MPICH_MAX_SHORT_MSG_SIZE. Note setting MPICH_UNEX_BUFFER_SIZE too large ( > 2 GB) may confuse MPICH and also lead to this message.

Go to Menu

# What does this mean? (continued)

- If you see this error message:

[0] MPIDI_PortalsU_Request_FDU_or_AEP: dropped event on unexpected receive queue, increase
[0] queue size by setting the environment variable MPICH_PTL_UNEX_EVENTS

- It means:

  You have exhausted the event queue entries associated with the unexpected queue. The default size is 20480.

- Try doing this to work around the problem:

  You can increase the size of this queue by setting the environment variable MPICH_PTL_UNEX_EVENTS to some value higher than 20480.

# What does this mean? (continued)

- If you see this error message:

[0] MPIDI_Portals_Progress: dropped event on "other" queue, increase

[0] queue size by setting the environment variable MPICH_PTL_OTHER_EVENTS

aborting job: Dropped Portals event

- It means:

  You have exhausted the event queue entries associated with the “other” queue. This can happen if the application is posting many non-blocking sends, or a large number of pre-posted receives are being posted, or many MPI-2 RMA operations are posted in a single epoch. The default size of the other EQ is 2048.

- Try doing this to work around the problem:

  You can increase the size of this queue by setting the environment variable MPICH_PTL_OTHER_EVENTS to some value higher than the 2048 default.

# What does this mean? (continued)

- If you see this error message:

0:(/notbackedup/users/rsrel/rs64.REL_1_3_12.051214.Wed/pe/compute

libs/mpich2/src/mpid/portals32/src/portals_progress.c:642)

PtlEQAlloc failed : PTL_NO_SPACE

- It means:

  You have requested so much EQ space for MPI (and possibly SHMEM if using both in same application) that there are not sufficient Portals resources to satisfy the request.

- Try doing this to work around the problem:

  You can decrease the size of the event queues by setting the environment variable MPICH_PTL_UNEXPECTED_EVENTS and MPICH_PTL_OTHER_EVENTS to smaller values.

# What does this mean? (continued)

- If you see this error message:

aborting job: Fatal error in MPI_Init: Other MPI error, error stack: MPIR_Init_thread(195): Initialization failed MPID_Init(170): failure during portals initialization MPIDI_Portals_Init(321): progress_init failed MPIDI_PortalsI_Progress_init(653): Out of memory

- It means:

  There is not enough memory on the nodes for the program plus MPI buffers to fit.

- Try doing this to work around the problem:

  You can decrease the amount of memory that MPI is using for buffers by using MPICH_UNEX_BUFFER_SIZE environment variable.

# What does this mean? (continued)

- If an MPI rank exits abnormally, ideally, Process Manager Interface (PMI) daemon reports the error

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

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

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

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

# What does this mean? (continued)

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

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

- Recommendation: For Cray MPI/Portals out-of-resources errors, given by a message, follow the advice

```
[193] MPICH PtlEQPoll error (PTL_EQ_DROPPED): An event was
dropped on the UNEX EQ handle.  Try increasing the value of
env var MPICH_PTL_UNEX_EVENTS (cur size is 20480).
```

# Outline: Cray MPI Communication Tips

- Cray MPI Point-to-Point Messaging Tips
- Cray MPI Collective Messaging Tips
- MPI Collectives: Memory Usage when Scaling Applications
- MPI Collectives: Memory Usage for MPI_Alltoall
- MPI Collectives: Memory Usage for Scaling MPI_Alltoall

# Cray MPI Point-to-Point Messaging Tips

- Pre-posting receives is generally a good idea (see the example on pp. 41-42)
  - For EAGER messages, this avoids an extra memcpy
  - Portals/Seastar handles the data copy directly into the user buffer
  - Can off-load work from CPU
  - Avoid posting thousands of receives
- Non-contiguous data types
  - More efficient to use contiguous data types for message transfers
  - If discontiguous, MPI must:
    - Send side: Allocate temp buffer, pack user data into temp buffer
    - Entire message is sent over network as contiguous
    - Recv side: Unpack temp buffer into user buffer
- Avoid “swamping” a busy rank with thousands of messages
  - Reduce MPICH_MAX_SHORT_MSG_SIZE to force rendezvous protocol
  - Consider enabling MPICH_PTL_SEND_CREDITS “flow-control” feature
  - Modify code to use explicit handshaking to minimize number of in-flight messages

# Cray MPI Collective Messaging Tips

- Cray Optimized Collectives
  - Work for any intra-communicator (not just MPI_COMM_WORLD)
  - Enabled by default
  - Many have user-adjustable cross-over points (see man page)
  - Can be selectively disabled via MPICH_COLL_OPT_OFF
    - **export MPICH_COLL_OPT_OFF=mpi_bcast,mpi_allgather**
- Cray MPI_Alltoallv / MPI_Alltoallw algorithm
  - Pairwise exchange with windows
  - Default window sizes set to allow 20 simultaneous sends/recvs
  - Set window sizes to 1 when scaling with medium/large messages
    - **export MPICH_ALLTOALLVW_SENDWIN=1**
    - **export MPICH_ALLTOALLVW_RECVWIN=1**
- Cray-Optimized SMP-aware Collectives
  - MPI_Allreduce
  - MPI_Barrier
  - MPI_Bcast ( new in MPT 3.1.1 )
  - MPI_Reduce ( new in MPT 3.1.2 )

Go to Menu

# MPI Collectives: Memory Usage when Scaling Applications

- Watch Memory Footprint per node as Applications Scale
  - Understand application memory usage as process count per node increases
  - MPI unexpected buffers the largest consumer for MPI internally
    - Default is 260MB per process for 150,000 rank job
    - Decrease by reducing size of MPICH_UNEX_BUFFER_SIZE
- MPI Collective Memory Usage
  - When scaling, watch use of collectives that accumulate data on a per-rank basis
  - MPI_Alltoall, MPI_Allgather, MPI_Gather, etc.
- Options to Decrease Memory Footprint
  - Decrease process density per node (-N8 vs –N6, –N4, –N2, –N1)
    - For more information on `aprun` command, please, refer to *A Guide to Using NCCS Jaguar System* at http://www.nccs.gov/user-support/training-education/hpcparallel-computing-links/ or man page by typing “man aprun”.
    - Specify aprun options to use both NUMA nodes on a socket
  - Consider hybrid MPI + OMP approach

# MPI Collectives: Memory Usage for MPI_Alltoall

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

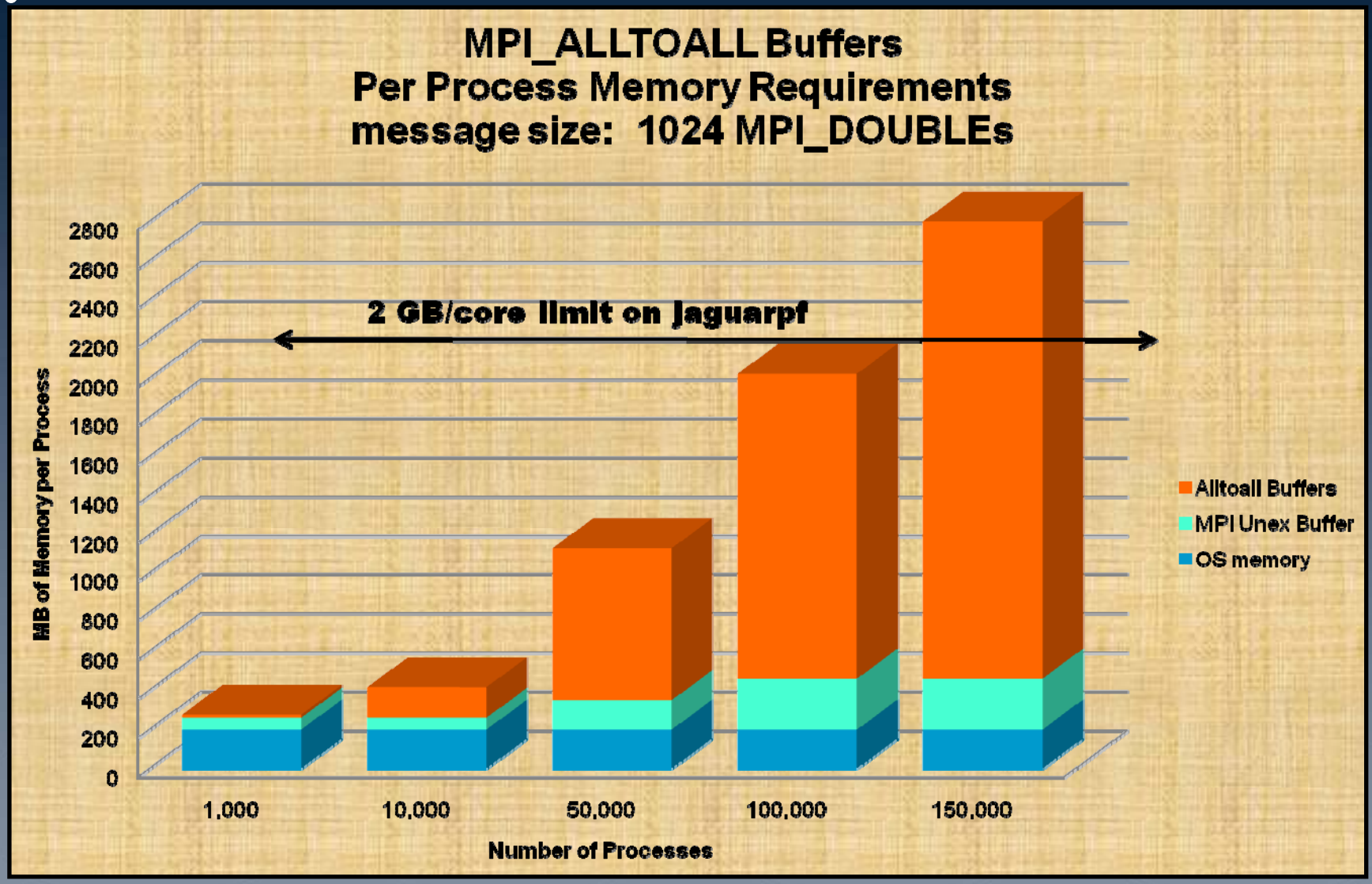
***Consider the following code snippet...***

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

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

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

# MPI Collectives: Memory Usage for Scaling MPI_Alltoall



# Outline: MPI Environment Variables

- Some MPI Environment Variable Tips
- MPICH_FAST_MEMCPY
- MPICH_COLL_SYNC
- MPICH_MPIIO_HINTS
- MPICH_MPIIO_HINTS (2)
- MPICH_MPIIO_CB_ALIGN
- MPICH_ENV_DISPLAY
- MPICH_SMP_OFF
- MPICH_PTL_MATCH_OFF and MPICH_PTL_SEND_CREDITS
- MPICH_PTL_MATCH_OFF

# Some MPI Environment Variables Tips

- Current version of MPT: 3.2.0 (Aug 2009)
- MPT attempts to set the right buffer sizes at launch time (rather than static settings)
- Suggestion: if you use env vars based on MPT 3.0 or earlier, comment them out and try out 3.2.0 without env vars.

# Some MPI Environment Variables Tips (continued)

- Next few slides cover environment variables that are associated with MPI performance
- Default settings are set based on the best performance on most codes.
  - Some codes may benefit from setting or adjusting environment variable settings.
- Find much of this information with “man mpi”
- As shown on previous slide, the MPI environment has changed significantly, thus it is important to re-read the MPI man pages and other related documents.

# MPI Environment Variables: MPICH_ENV_DISPLAY

- If set, causes rank 0 to display all Cray MPI/MPICH environment variables and their current settings at MPI initialization time.
- Default: Not enabled.
- Useful for debugging purposes.
- MPICH_VERSION_DISPLAY - displays the version of Cray MPT being used

# MPI Environment Variables: MPICH_FAST_MEMCPY

- If set, enables an optimized memcpy routine in MPI. The routine is used for node-local memory copies in the point-to-point and collective MPI operations.
  - This can increase the performance of some collectives that send large (256K and greater) messages.
    - Collectives are almost always faster
    - Speedup varies by message size
  - By default - is not enabled. There are few cases where it causes performance degradation
  - To enable, `export MPICH_FAST_MEMCPY=1`

# MPI Environment Variables: MPICH_COLL_SYNC

- If set, a Barrier is performed at the beginning of each specified MPI collective function. This forces all processes participating in that collective to sync up before the collective communication can begin.
  - To enable this feature for all MPI collectives, set the value to 1. *Default is off.*
- Can be enabled for a selected list of MPI collectives
- There are *rare* cases where this variable helps
  - If a code has lots of collectives and MPI profiling shows imbalance (sync time is an issue), this *may* help

# MPI Environment Variables: MPICH_MPIIO_HINTS

- If set, override the default value of one or more MPI-IO hints. This also overrides any value set in the application code with calls to the MPI_Info_set routine.
- The hints are applied to the file when it is opened with an MPI_File_open() call.
- MPICH_MPIIO_HINTS_DISPLAY
  - If set, causes rank 0 in the participating communicator to display the names and values of all MPI-IO hints that are set for the file being opened with the MPI_File_open call.

Default settings:

```
PE 0: MPIIO hints for
 c2F.TILT3d.hdf5:
  cb_buffer_size     = 16777216
  romio_cb_read      = automatic
  romio_cb_write     = automatic
  cb_nodes           = #nodes/8
  romio_no_indep_rw  = false
  ind_rd_buffer_size= 4194304
  ind_wr_buffer_size= 524288
  romio_ds_read      = automatic
  romio_ds_write     = automatic
  direct_io          = false
  cb_config_list     = *:1
```

# MPI Environment Variables: MPICH_MPIIO_HINTS (continued)

Examples:

- Syntax
  - `export MPICH_MPIIO_HINTS=data.hdf5:direct_io=true`
- For FlashIO at 5000 processes writing out 500MB per MPI thread, the following improved performance:

```
romio_cb_write = “ENABLE”
romio_cb_read = “ENABLE”
cb_buffer_size = 32M
```

  - When enabled, all collective reads/writes will use collective buffering. When disabled, all collective reads/writes will be serviced with individual operations by each process. When set to automatic, ROMIO will use heuristics to determine when to enable the optimization.
- For S3D at 10K cores:

`romio_ds_write = ‘disable’` - specifies if data sieving is to be done on read. Data sieving is a technique for efficiently accessing noncontiguous regions of data

`romio_no_indep_rw = ‘true’` - specifies whether deferred open is used.

  - Romio docs say that this indicates no independent read or write operations will be performed. This can be used to limit the number of processes that open the file.

# MPI Environment Variables: MPICH_MPIIO_CB_ALIGN

- If set to 1, new algorithms that take into account physical I/O boundaries and the size of I/O requests are used to determine how to divide the I/O workload when collective buffering is enabled.
  - This can improve performance by causing the I/O requests of each collective buffering node (aggregator) to start and end on physical I/O boundaries and by preventing more than one aggregator making reference to any given stripe on a single collective I/O call.
  - If set to zero or not defined, the algorithms used prior to MPT release 3.1 are used.
  - Default: not set

# MPI Environment Variables: MPICH_SMP_OFF

- If set, disable the on-node SMP device and use the Portals device for all MPI message transfers
- Use in a rare cases where code benefits from using Portals matching instead of MPI matching.
- Default: Not enabled.
- Useful for debugging reproducibility issues.

# MPICH_PTL_MATCH_OFF and MPICH_PTL_SEND_CREDITS

MPICH_PTL_MATCH_OFF

- If set, disables registration of receive requests with portals.
  - Setting this allows MPI to perform the message matching for the portals device. It may be beneficial to set this variable when an application exhausts portals internal resources and for latency-sensitive applications.

MPICH_PTL_SEND_CREDITS

- Enables flow control to prevent the Portals event queue from being overflowed.
  - Value of '-1' should prevent queue overflow in any situation
  - Should only be used as needed, as flow control will result in less optimal performing code. If the Portals unexpected event queue can not be increased enough, then flow control may need to be enabled.

# MPI Environment Variable: MPICH_PTL_MATCH_OFF

- Case where MPICH_PTL_MATCH_OFF fixed an MPI problem

  [3683] : (/tmp/ulib/mpt/nightly/3.0/042108/xt/trunk/mpich2/src/
  mpid/cray/src/adi/ptldev.c:2693)

  PtlMEMDPost() failed : PTL_NO_SPACE

- For this, try MATCH, OTHER_EVENTS or SEND_CREDITS env var

  [43] MPICH PtlEQPoll error (PTL_EQ_DROPPED): An event was dropped on the OTHER EQ handle. Try increasing the value of env var MPICH_PTL_OTHER_EVENTS (cur size is 2048).

  aborting job:

  PtlEQPoll/PtlEQGet error

  – Attempts to increase OTHER_EVENTS did not help though (in this case)

# Outline: MPI Rank Placement

- Rank Placement Tips
- Rank order and CrayPAT
- Reordering Workflow
- CrayPAT example

# Rank Placement Tips

- In some cases changing how the processes are laid out on the machine may affect performance, by relieving synchronization/imbalance time.
- The default is currently SMP-style placement. This means that for or a multi-node core, sequential MPI ranks are placed on the same node.
  - In general, MPI codes perform better using SMP placement
  - Collectives have been optimized to be SMP aware
- For example, an 12-process job launched on a XT5 node with two 6-core processors would be placed as:

  PROCESSOR 0, 1

  RANK 0,1,2,3 4,5,6,7,8,9,10,11,12

# Rank Placement Tips (continued)

- 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 MPI ranks are placed on the next node in the list.

1: SMP-style placement. All cores from all nodes are allocated in a sequential order.

2: Folded rank placement. Similar to default ordering except that the tasks N+1 ... 2N are mapped to slave cores of nodes N ... 1.

3: Custom ordering. The ordering is specified in a file named MPICH_RANK_ORDER.

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

# Rank order and CrayPAT

- One can also use the CrayPat performance measurement tools to generate a suggested custom ordering.
  - Available if MPI functions are traced (-g mpi or –O apa)
  - `pat_build -O apa my_program`
    - see Examples section of pat_build man page
- `pat_report` options:
  - `mpi_sm_rank_order`
    - Uses message data from tracing MPI to generate suggested MPI rank order. Requires the program to be instrumented using the pat_build -g mpi option.
  - `mpi_rank_order`
    - Uses time in user functions, or alternatively, any other metric specified by using the -s mro_metric options, to generate suggested MPI rank order.

# Using CrayPat: Reordering Workflow

- module load xt-craypat/4.4.1
- Rebuild your code
- pat_build –O apa a.out
- Run a.out+pat
- pat_report –Ompi_sm_rank_order a.out+pat+…sdt/ > pat.report
- Creates MPICH_RANK_REORDER_METHOD.x file
- Then set env var MPICH_RANK_REORDER_METHOD=3 AND
- Link the file MPICH_RANK_ORDER.x to MPICH_RANK_ORDER
- Rerun code

# CrayPAT example

```
Table 1: Suggested MPI Rank Order

 Eight cores per node: USER Samp per node
 Rank          Max     Max/          Avg     Avg/  Max Node
Order   USER Samp      SMP   USER Samp      SMP  Ranks
    d       17062    97.6%       16907  100.0%  832,328,820,797,113,478,898,600
    2       17213    98.4%       16907  100.0%  53,202,309,458,565,714,821,970
    0       17282    98.8%       16907  100.0%  53,181,309,437,565,693,821,949
    1       17489   100.0%       16907  100.0%  0,1,2,3,4,5,6,7
```

- This suggests that
  1. the custom ordering “d” might be the best
  2. Folded-rank next best
  3. Round-robin 3rd best
  4. Default ordering last

# Outline: MPI Programming Techniques

- Pre-posting receives
- Overlapping communication with computation
- Example: 9-pt stencil pseudo-code
- Example: 9-pt stencil update
- Aggregating data
- Aggregating data: Example from CFD

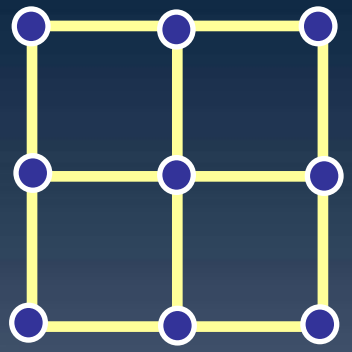
# Pre-posting receives

- If possible, pre-post receives before sender posts the matching send
  - Optimization - typically useful technique for all MPICH installations
- But be careful with excessive pre-posting of the receives though, as it will hit Portals internal resource limitations eventually (see slide 15 for resource limits).

# Overlapping communication with computation

- The basic idea of overlapping communication with computation (though not all systems support it) is that one initiates communication and, while data is being transferred, does some computation instead of waiting. Ideally, by the time the computation is done the communication is also done.
- Highly algorithmic dependent - you can't overlap everything!
- This is a corollary of pre-posting receives because using “isends” and “irecvs” is a common way of trying to do overlapped communication and computation.
- Use non-blocking send/recv calls when it is possible to overlap communication with computation.
- In some cases it may be better to replace collective operations with point to point communications in order to overlap communication with computation.
  - Caution: Not suggesting every collective be re-programmed by hand
  - It may be that a certain part of your algorithm has computation that could overlap the point to point communications that would not happen with a [blocking] collective.
- For more information search the internet on “Overlapping communication with computation”

# Example: 9-pt stencil pseudo-code



9-point stencil

Basic:

9 pt computation
Update ghost cell boundaries
East/West IRECV, ISEND, WAITALL
North/South IRECV, ISEND, WAITALL

Maximal Irecv preposting:

Prepost all IRECV
9 pt computation
Update ghost cell boundaries
East/West ISEND,
Wait on E/W IRECV only
North/South ISEND,
Wait on the rest

# Example: 9-pt stencil update

```
!compute stencil
…

!update ghost cell boundaries.
!East/West
MPI_IRECV(XOUT(1,1), 1, mpi_ew_type, nbr_west,
   mpitag_wshift, COMM_OCN, request(3))
MPI_IRECV(XOUT(iphys_e+1,1), 1, mpi_ew_type,
   nbr_east, mpitag_eshift, COMM_OCN,
   request(4))
MPI_ISEND(XOUT(iphys_e+1-num_ghost_cells,1),
   1, mpi_ew_type, nbr_east, mpitag_wshift,
   COMM_OCN, request(1))
MPI_ISEND(XOUT(iphys_b,1), 1, mpi_ew_type,
   nbr_west, mpitag_eshift, COMM_OCN,
   request(2))
MPI_WAITALL(4, request, status)

!North/South
MPI_IRECV(XOUT(1,jphys_e+1), 1, mpi_ns_type,
   nbr_north, mpitag_nshift, COMM_OCN,
   request(3))
MPI_IRECV(XOUT(1,1), 1, mpi_ns_type,
   nbr_south, mpitag_sshift, COMM_OCN,
   request(4))
MPI_ISEND(XOUT(1,jphys_b), 1, mpi_ns_type,
   nbr_south, mpitag_nshift, COMM_OCN,
   request(1))
MPI_ISEND(XOUT(1,jphys_e+1-num_ghost_cells),
   1, mpi_ns_type, nbr_north, mpitag_sshift,
   COMM_OCN, request(2))
MPI_WAITALL(4, request, status)
```

```
! Prepost receive requests
MPI_IRECV(buf_west_rcv, buf_len_ew,
   MPI_DOUBLE_PRECISION, nbr_west, &
   mpitag_wshift, COMM_OCN, request(7))
MPI_IRECV(buf_east_rcv, buf_len_ew,
   MPI_DOUBLE_PRECISION, nbr_east,
   mpitag_eshift, COMM_OCN, request(8))
MPI_IRECV(XOUT(1,jphys_e+1), buf_len_ns,
   MPI_DOUBLE_PRECISION, nbr_north,
   mpitag_nshift, COMM_OCN, request(5))
MPI_IRECV(XOUT(1,1), buf_len_ns,
   MPI_DOUBLE_PRECISION, nbr_south,
   mpitag_sshift, COMM_OCN, request(6))
! compute stencil
…
! send east-west boundary info
MPI_ISEND(buf_east_snd, buf_len_ew,
   MPI_DOUBLE_PRECISION, nbr_east,
   mpitag_wshift, COMM_OCN, request(1))
MPI_ISEND(buf_west_snd, buf_len_ew,
   MPI_DOUBLE_PRECISION, nbr_west,
   mpitag_eshift, COMM_OCN, request(2))
MPI_WAITALL(2, request(7), status_wait)
! send north-south boundary info
MPI_ISEND(XOUT(1,jphys_e+1-num_ghost_cells),
   buf_len_ns, MPI_DOUBLE_PRECISION,
   nbr_north, mpitag_sshift, COMM_OCN,
   request(3))
MPI_ISEND(XOUT(1,jphys_b), buf_len_ns,
   MPI_DOUBLE_PRECISION, nbr_south,
   mpitag_nshift, COMM_OCN, request(4))
MPI_WAITALL(6, request, status_wait)
```

Go to Menu

# Aggregating data

- For very small buffers, aggregate data into fewer MPI calls (especially for collectives)
  - Ex. 1 alltoall with an array of 3 reals is clearly better than 3 alltoalls with 1 real
  - Do not aggregate too much. The MPI protocol switches from an short (eager) protocol to a long message protocol using a receiver pull method once the message is larger than the eager limit. This limit is by default 128000 bytes, but it can be changed with the MPICH_MAX_SHORT_MSG_SIZE environment variable. The optimal size for messages most of the time is less than the eager limit.

# Aggregating data: Example from CFD

*****Original*****

```
for (index = 0; index < No; index++){
    double tmp;
    tmp = 0.0;
    out_area[index] = Bndry_Area_out(A,
    labels[index]);
    gdsum(&outlet_area[index],1,&tmp);
}
for (index = 0; index < Ni; index++){
    double tmp;
    tmp = 0.0;
    in_area[index] = Bndry_Area_in(A,
    labels[index]);
    gdsum(&inlet_area[index],1,&tmp);
}
void gdsum (double *x,int n,double *work)
{
    register int i;
    MPI_Allreduce (x, work, n,
    MPI_DOUBLE, MPI_SUM, MPI_COMM_WORLD);
    /* *x = *work; */
    dcopy(n,work,1,x,1);
    return;
}
```

*****Improved*****

```
for (index = 0; index < No; index++)
  {out_area[index] = Bndry_Area_out(A,
  labels[index]);
  }
/* Get gdsum out of for loop */
tmp = new double[No];
gdsum (outlet_area, No, tmp);
delete tmp;
for (index = 0; index < Nin; index++)
  {in_area[index] = Bndry_Area_in(A,
  labels[index]);
  }
/* Get gdsum out of for loop */
tmp = new double[Ni];
gdsum(inlet_area, Ni, tmp);
delete tmp;
```

# Outline: Resources for Users

- man pages and MPI web-sites
- MPI Books
- Getting Started
- Advanced Topics
- More Information

# Resources for Users: man pages and MPI web-sites

- There are man pages available for MPI which should be installed in your MANPATH. The following man pages have some introductory information about MPI.

```
% man MPI
% man cc
% man ftn
% man qsub
% man MPI_Init
% man MPI_Finalize
```

- MPI man pages are also available online. http://www.mcs.anl.gov/mpi/www/
- Main MPI web page at Argonne National Laboratory http://www-unix.mcs.anl.gov/mpi
- Set of guided exercises http://www-unix.mcs.anl.gov/mpi/tutorial/mpiexmpl
- MPI tutorial at Lawrence Livermore National Laboratory https://computing.llnl.gov/tutorials/mpi/
- MPI Forum home page contains the official copies of the MPI standard. http://www.mpi-forum.org/

# Resources for Users: MPI Books

- Books on and about MPI
  - Using MPI, 2nd Edition by William Gropp, Ewing Lusk, and Anthony Skjellum, published by MIT Press ISBN 0-262-57132-3. The example programs from this book are available at ftp://ftp.mcs.anl.gov/pub/mpi/using/UsingMPI.tar.gz.
    The Table of Contents is also available. An errata for the book is available. Information on the first edition of Using MPI is also available, including the errata. Also of interest may be The LAM companion to ``Using MPI..." by Zdzislaw Meglicki (gustav@arp.anu.edu.au).
  - Designing and Building Parallel Programs is Ian Foster's online book that includes a chapter on MPI. It provides a succinct introduction to an MPI subset. (ISBN 0-201-57594-9; Published by Addison-Wesley>)
  - **MPI: The Complete Reference**, by Marc Snir, Steve Otto, Steven Huss-Lederman, David Walker, and Jack Dongarra, The MIT Press .
  - MPI: The Complete Reference - 2nd Edition: Volume 2 - The MPI-2 Extensions, by William Gropp, Steven Huss-Lederman, Andrew Lumsdaine, Ewing Lusk, Bill Nitzberg, William Saphir, and Marc Snir, The MIT Press.
  - Parallel Programming With MPI, by Peter S. Pacheco, published by Morgan Kaufmann.
  - RS/6000 SP: Practical MPI Programming, by Yukiya Aoyama and Jun Nakano (IBM Japan), and available as an IBM Redbook.
  - **Supercomputing Simplified: The Bare Necessities for Parallel C Programming with MPI**, by William B. Levy and Andrew G. Howe, ISBN: 978-0-9802-4210-2. See the website for more information.

# Resources for Users: Getting Started

- About Jaguar

  http://www.nccs.gov/computing-resources/jaguar/

- Quad Core AMD Opteron Processor Overview

  http://www.nccs.gov/wp-content/uploads/2008/04/amd_craywkshp_apr2008.pdf

- PGI Compilers for XT5

  http://www.nccs.gov/wp-content/uploads/2008/04/compilers.ppt

- NCCS Training & Education – archives of NCCS workshops and seminar series, HPC/parallel computing references

  http://www.nccs.gov/user-support/training-education/

- 2009 Cray XT5 Quad-core Workshop

  http://www.nccs.gov/user-support/training-education/workshops/2008-cray-xt5-quad-core-workshop/

# Resources for Users: Advanced Topics

- Debugging Applications Using TotalView

  http://www.nccs.gov/user-support/general-support/software/totalview

- Using Cray Performance Tools - CrayPat

  http://www.nccs.gov/computing-resources/jaguar/debugging-optimization/cray-pat/

- I/O Tips for Cray XT4

  http://www.nccs.gov/computing-resources/jaguar/debugging-optimization/io-tips/

- NCCS Software

  http://www.nccs.gov/computing-resources/jaguar/software/

# Resources for Users: More Information

- NCCS website

  http://www.nccs.gov/

- Cray Documentation

  http://docs.cray.com/

- Contact us

  help@nccs.gov