CRAY

Tips When Using Cray MPI

Legal Disclaimer

Information in this document is provided in connection with Cray Inc. products. No license, express or implied, to any intellectual property rights is granted by this document.

Cray Inc. may make changes to specifications and product descriptions at any time, without notice.

All products, dates and figures specified are preliminary based on current expectations, and are subject to change without notice.

Cray hardware and software products may contain design defects or errors known as errata, which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Cray uses codenames internally to identify products that are in development and not yet publically announced for release. Customers and other third parties are not authorized by Cray Inc. to use codenames in advertising, promotion or marketing and any use of Cray Inc. internal codenames is at the sole risk of the user.

Performance tests and ratings are measured using specific systems and/or components and reflect the approximate performance of Cray Inc. products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance.

The following are trademarks of Cray Inc. and are registered in the United States and other countries: CRAY and design, SONEXION, URIKA, and YARCDATA. The following are trademarks of Cray Inc.: ACE, APPRENTICE2, CHAPEL, CLUSTER CONNECT, CRAYPAT, CRAYPORT, ECOPHLEX, LIBSCI, NODEKARE, THREADSTORM. The following system family marks, and associated model number marks, are trademarks of Cray Inc.: CS, CX, XC, XE, XK, XMT, and XT. The registered trademark LINUX is used pursuant to a sublicense from LMI, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis. Other trademarks used in this document are the property of their respective owners.

Agenda

- Introduction to Cray MPI
- Tips and useful environment variables for Cray systems with Intel Xeon and Xeon-phi processors
- Hybrid MPI + OpenMP applications
- General Recommendations

Brief Introduction to Cray MPI

- **I/O, collectives, P2P, and one-sided all optimized for Cray system architecture**
  - SMP-aware collectives
  - High performance single-copy on-node communication via xpmem (not necessary to program for shared memory)

- **Highly tunable through environment variables**
  - Defaults should generally be best, but some cases benefit from fine tuning

- **Integrated within the Cray Programming Environment**
  - Compiler drivers manage compile flags and linking automatically
  - Profiling through Cray performance tools

Cray MPI Documentation

- **Primary user resource for tuning and feature documentation is the man page**
  - man intro_mpi

  OR

  - man MPI

- **Standard function documentation available as well**
  - E.g., man mpi_isend

MPI Rank Reorder – Two Interfaces Available

- **CrayPat**
  - Available with sampling or tracing
  - Include –g mpi when instrumenting program
  - Run program and let CrayPat determine if communication is dominant, detect communication pattern and suggest MPI rank order if applicable

- **grid_order utility**
  - User knows communication pattern in application and wants to quickly create a new MPI rank placement file
  - Available when perftools-base module is loaded

MPI Rank Order Observations

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

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

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

MPI Rank Order Observations (2)

```
MPI Grid Detection:

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

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

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

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

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/W
ORKSHOP/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.
#
0,532,64,564,32,572,96,540,8,5
96,72,524,40,604,24,588
104,556,16,628,80,636,56,620,4
8,516,112,580,88,548,120,612
1,403,65,435,33,411,97,443,9,4
67,25,499,105,507,41,475
73,395,81,427,57,459,17,419,11
3,491,49,387,89,451,121,483
6,436,102,468,70,404,38,412,14
,444,46,476,110,508,78,500
86,396,30,428,62,460,54,492,11
8,420,22,452,94,388,126,484
129,563,193,531,161,571,225,53
9,241,595,233,523,249,603,185,
555
153,587,169,627,137,635,201,61
9,177,515,145,579,209,547,217,
611
7,405,71,469,39,437,103,413,47
,445,15,509,79,477,31,501
111,397,63,461,55,429,87,421,2
3,493,119,389,95,453,127,485
134,402,198,434,166,410,230,44
2,238,466,174,506,158,394,246,
474
190,498,254,426,142,458,150,38
6,182,418,206,490,214,450,222,
482
128,533,192,541,160,565,232,52
5,224,573,240,597,184,557,248,
605
168,589,200,517,152,629,136,54
9,176,637,144,621,208,581,216,
613
5,439,37,407,69,447,101,415,13
,471,45,503,29,479,77,511
53,399,85,431,21,463,61,391,10
9,423,93,455,117,495,125,487
2,530,34,562,66,538,98,522,10,
570,42,554,26,594,50,602
18,514,74,586,58,626,82,546,10
6,634,90,578,114,618,122,610
135,315,167,339,199,347,259,30
7,231,371,239,379,191,331,247,
299
175,363,159,323,143,355,255,29
1,207,275,183,283,151,267,215,
223
133,406,197,438,165,470,229,41
4,245,446,141,478,237,502,253,
398
157,510,189,462,173,430,205,39
0,149,422,213,454,181,494,221,
486
130,316,260,340,194,372,162,34
8,226,308,234,380,242,332,250,
300
202,364,186,324,154,356,138,29
2,170,276,178,284,210,218,268,
146
4,535,36,543,68,567,100,527,12
,599,44,575,28,559,76,607
52,591,20,631,60,639,84,519,10
8,623,92,551,116,583,124,615
3,440,35,432,67,400,99,408,11,
464,43,496,27,472,51,504
19,392,75,424,59,456,83,384,10
7,416,91,488,115,448,123,480
132,401,196,441,164,409,228,43
3,236,465,204,473,244,393,188,
497
252,505,140,425,212,457,156,38
5,172,417,180,449,148,489,220,
481
131,534,195,542,163,566,227,52
6,235,574,203,598,243,558,187,
606
251,590,211,630,179,638,139,62
2,155,550,171,518,219,582,147,
614
761,660,737,652,705,668,745,69
2,673,700,641,684,713,644,753,
724
729,732,681,756,721,716,764,67
6,697,748,689,657,740,665,649,
708
760,528,736,536,704,560,744,52
0,672,568,712,592,752,552,640,
600
728,584,680,624,720,512,696,63
2,688,616,664,544,608,656,648,
576
762,659,738,651,706,667,746,64
3,714,691,674,699,754,683,730,
723
722,731,763,658,642,755,739,67
5,707,650,682,715,698,666,690,
747
257,345,265,313,281,305,273,33
7,609,369,577,377,617,329,513,
529
545,297,633,361,625,321,585,53
7,601,289,553,353,593,521,569,
561
256,373,261,341,264,349,280,31
7,272,381,269,309,285,333,277,
365
352,301,320,325,288,357,328,30
4,360,312,376,293,296,368,336,
344
258,338,266,346,282,314,274,37
0,766,306,710,378,742,330,678,
362
646,298,750,322,718,354,758,29
0,734,662,686,670,726,702,694,
654
262,375,263,343,270,311,271,35
1,286,319,278,342,287,350,279,
374
294,318,358,383,359,310,295,38
2,326,303,327,367,366,335,302,
334
765,661,709,663,741,653,711,66
9,767,655,743,671,749,695,679,
703
677,727,751,693,647,701,717,68
7,757,685,733,725,719,735,645,
759
```

MPICH_RANK_REORDER_METHOD

- Vary your rank placement to optimize communication
- Can be a quick, low-hassle way to improve performance
- Use CrayPAT to produce a specific MPICH_RANK_ORDER file to maximize intra-node communication
- Or, use grid_order utility with your application's grid dimensions to layout MPI ranks in alignment with data grid
- To use:
  - name your custom rank order file: MPICH_RANK_ORDER
  - export MPICH_RANK_REORDER_METHOD=3

MPI Rank Reorder (continued)

- A topology and placement-aware reordering method is also available (uses node allocation information)
- Optimizes rank ordering for Cartesian decompositions using the layout of nodes in the job
- To use:
  - `user@login> export MPICH_RANK_REORDER_METHOD=4`
  - `user@login> export MPICH_RANK_REORDER_OPTS=\`
    `"-ndims=3 -dims=16,16,8"`
  - See intro_mpi(1) man page for more information

HUGEPAGES

- Linking and running with hugepages can offer a significant performance improvement for many MPI communication sequences, including MPI collectives and basic MPI_Send / MPI_Recv calls

- Most important for applications calling MPI_Alltoall[v] or performing point-to-point operations with a similarly well connected pattern

- To use HUGEPAGES, load desired module at link and run time:
  - `module load craype-hugepages8M` (many sizes supported)
  - << *re-link your app* >>
  - `module load craype-hugepages8M`
  - << *run your app* >>

Using DMAPP

- **DMAPP optimizations not enabled by default because…**
  - May reduce resources MPICH has available (shared with DMAPP)
  - Requires more memory (for DMAPP internals)
  - DMAPP does not handle transient network errors

- **These are highly-optimized algorithms which may result in significant performance gains, but user has to request them**

- **Supported DMAPP-optimized functions**
  - MPI_Allreduce (4-8 bytes)
  - MPI_Bcast (4 or 8 bytes)
  - MPI_Barrier
  - MPI_Put / MPI_Get / MPI_Accumulate

- **To use, link with `libdmapp` and set the following environment variable**
  - Collective use: `user@login>` export MPICH_USE_DMAPP_COLL=1
  - RMA one-sided use: `user@login>` export MPICH_RMA_OVER_DMAPP=1

MPICH GNI Environment Variables

Used to optimize inter-node traffic using the Aries interconnect, the following are the most significant variables to try (*avoid significant deviations from the default if possible*):

- **MPICH_GNI_MAX_VSHORT_MSG_SIZE**
  - Controls max message size for E0 mailbox path (Default: varies)
- **MPICH_GNI_MAX_EAGER_MSG_SIZE**
  - Controls max message size for E1 Eager Path (Default: 8K bytes)
- **MPICH_GNI_NUM_BUFS**
  - Controls number of 32KB internal buffers for E1 path (Default: 64)
- **MPICH_GNI_NDREG_MAXSIZE**
  - Controls max message size for R0 Rendezvous Path (Default: 4MB)
- **MPICH_GNI_RDMA_THRESHOLD**
  - Controls threshold for switching to BTE from FMA (Default: 1K bytes)

See the MPI man page for further details

Specific Collective Algorithm Tuning

- **Different algorithms may be used for different message sizes in collectives (e.g.)**
  - Algorithm A might be used for Alltoall for messages < 1K
  - Algorithm B might be used for messages >= 1K

- **To optimize a collective, you can modify the cutoff points when different algorithms are used, which may improve performance**

- **MPICH_ALLGATHER_VSHORT_MSG**
- **MPICH_ALLGATHERV_VSHORT_MSG**
- **MPICH_GATHERV_SHORT_MSG**
- **MPICH_SCATTERV_SHORT_MSG**
- **MPICH_GNI_A2A_BLK_SIZE**
- **MPICH_GNI_A2A_BTE_THRESHOLD**
- **MPICH_MAX_THREAD_SAFETY=multiple (for thread multiple support)**

See the MPI man page for further details

Using MPI + OpenMP

MPI Thread Multiple Support

- **Thread multiple support for**
  - point to point operations (optimized global lock)
  - Collectives (optimized global lock)
  - MPI-RMA (thread hot)

- **All supported in default library**

- **`user@login> export MPICH_MAX_THREAD_SAFETY=multiple`**

- **Global lock optimization on by default (N/A for MPI-RMA)**
  - 50% better 8B latency than pthread_mutex() (OSU latency_mt, 32 threads per node, Broadwell)
  - export MPICH_OPT_THREAD_SYNC=0 falls back to pthread_mutex()

Thread Hot Communication

"Thread hot": high performance thread multiple support

- Design Objectives
  - Contention Free progress and completion
  - High bandwidth and high message rate
  - Independent progress – thread(s) flush outstanding traffic, other threads make uninterrupted progress
  - Dynamic mapping between threads and network resources
  - Locks needed only if the number of threads exceed the number of network resources

- MPI-3 RMA
  - Epoch calls (`Win_complete, Win_fence`) are thread-safe, but not intended to be thread hot
  - All other RMA calls (including request-based operations) are thread hot
  - **Multiple threads doing Passive Synchronization operations likely to perform best**

Multi-threading Optimizations in Cray MPI

- **Easy way to hit the ground running on a KNL – MPI only mode**
  - Works quite well in our experience
  - Scaling to more than 2-8 threads most likely requires a different application design approach
- **“Bottom-Up” OpenMP development approach**
- **“Top-Down” SPMD model**
  - Increases the scope of code executed by OpenMP, allows for better load balancing and overall compute scaling on KNL
  - Allows multiple threads to call MPI concurrently
  - In this model, performance is limited by the level of support offered by MPI for multi-threaded communication
  - MPI implementations must offer “Thread-Hot” communication capabilities to improve communication performance for highly threaded use cases on KNL

Examples of MPI + OpenMP Use

“bottom up”

```
! Keep OpenMP within a “compute” loop

DO WHILE (t .LT. tend)

   DO patch = 1, npatches

      CALL update_patch()

      …CALL MPI…

   END DO

END DO

SUBROUTINE update_patch()

   !$OMP PARALLEL DO
   DO i = 1, nx
   …do work…
   END DO

END SUBROUTINE
```

“SPMD”

```
! Move OpenMP near the top of the call stack

!#OMP PARALLEL
DO WHILE (t .LT. tend)

   !#OMP DO
   DO patch = 1, npatches

      CALL update_patch()

      …CALL MPI…

   END DO

END DO
```

Recommendations

- **Using hugepages in MCDRAM can improve large message communication performance**
- **Using thread multiple with Cray's Thread-Hot capabilities on Intel Xeon and Intel KNL architectures is a key tool for hybrid applications**
- **Using asynchronous communication can hide/overlap communication overheads and improve application scalability**
- **MPI-only works quite well on KNL**
  - Threading can be helpful, but unless SPMD with "thread-hot" MPI is used scaling to more than 2-8 threads not recommended
- **Collectives implemented with user pt2pt is strongly discouraged**
  - **Especially for alltoall, bcast, and gather**
  - **Very unlikely pt2pt will perform better**
  - **If they do, please file a bug with Cray**