Managing Cray XT MPI Runtime Environment Variables to Optimize and Scale Applications

Geir Johansen, *Cray Inc.*

ABSTRACT: *The Cray XT implementation of MPI provides configurable runtime environment variables that can be used to further optimize the performance of an application. The environment variables are also used to configure MPI event queues and message buffers to increase the scalability of the application. The paper will outline the available MPI environment variables and how they have been used to optimize and scale applications.*

KEYWORDS: Cray XT3, Cray XT4, Programming Environment, MPI

1.0 Introduction

The goal of this paper is provide guidance to application analysts when porting MPI applications to the Cray XT system. Specifically, the paper will discuss the use of MPI runtime environment variables and how these variables can be used to further optimize and increase the scale of the application. The intent of the paper is not to be a reference for each of the 35 available MPI environment variables, but rather to share experiences of how these variables have been used.

The paper uses the Cray Message Passing Toolkit (MPT) 3.0 version of MPI to describe the options available to optimize and scale a MPI application. Differences between the MPT 3.0 environment variables and earlier MPT releases will be discussed. Also, MPI environment variable features introduced in MPT 3.0 will be presented. An important caveat to this paper is that it represents a current snapshot and the information provided is subject to change.

2.0 Cray XT MPI Implementation

The Cray Message Passing Toolkit (MPT) release contains the Cray implementation of MPI and SHMEM. The version of MPI released in MPT 3.0 is based on MPICH2 1.0.4p from Argonne National Laboratory. The Cray implementation of MPI supports most of the functionality of the MPI-2 standard. One feature of MPI-2 that is not currently supported in Cray MPI is the dynamic process creation and management feature.

Cray XT implementation of MPI supports the Portals abstract device interfaces (ADI3) to perform message communication between nodes. MPT 3.0 has added the support of the SMP ADI3 to perform on-node MPI communication. MPT 3.0 is supported on the Cray XT CNL system, but does not run on Cray XT Catamount systems. Cray MPI supports running the Portals and SMP devices concurrently. Each process of an MPI job will automatically choose the most optimal messaging path to every other process in the job.

2.1 Portals Abstract Device Interface

Portals is a software interface for communication between nodes of the Cray XT system. The main design focus of Portals was to perform MPI communication. In this section, a very brief description will be given of how Portals handles MPI communication. This information is useful in understanding how the Portals MPI environment variables are used. More details on the Cray MPI implementation of the Portals abstract device interface can be found in reference [2].

Portals uses an *eager* protocol for sending short messages. The size of a short message can be configured by the user and has a default message size of 128000 bytes. The sender in the *short message eager* protocol sends the message and assumes the receiver has sufficient resources to receive the message. If the message to be sent is 1024 bytes or less, the message is considered to be a *vshort* (very short) message. The handling of *vshort* messages allows blocking send calls to return to the sender more quickly.

On the receive side, if a receive has been pre-posted for the message, then the data will be transferred to the application's buffer allocated for the message. If a receive for the incoming message has not been pre-posted, the data will be placed in the *unexpected buffer* and two entries (one to indicate the start of message and another to indicate that the data transfer has completed) will be placed in the *unexpected event queue*. When the receiver has posted a receive for this message, the data will be copied out of the *unexpected buffer* and the entries in the *unexpected event queue* will be cleared.

Portals has two methods for sending long messages. The default method is the *receiver-pull long message* protocol. In this method the sender will notify the intended receiver that is has long message to send. When the receiver is ready to accept the data, it will send a request to *get* the data. The other method is the *eager long message* protocol. This method is similar to the *short message eager* protocol, except the receiver will not store incoming data if a receive has not been pre-posted. This method should only be used if the application can ensure that receives will be pre-posted.

2.2 SMP Abstract Device Interface

The SMP abstract device interface was introduced in MPT 3.0 to provide communication for on-node messaging. The SMP device provides much faster latency and higher bandwidth than the Portals device. As a result, most codes running on a multi-core Cray XT system will likely see a performance gain when switching from MPT 2.0 to MPT 3.0. The graph in figure 1 shows an example of the effect of the intra-node latency improvements when using the MPT 3.0 SMP device versus the MPT 2.0 Portals device.

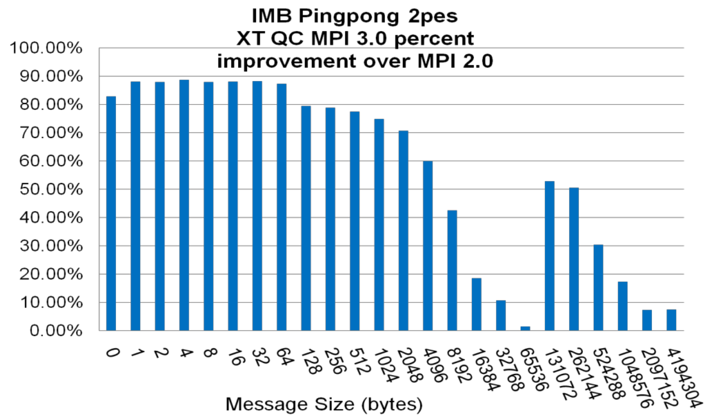

Figure 1. MPT 3.0 Latency Improvement Example

In MPT 3.0, the MPI collectives have been optimized to be SMP aware. Figure 2 shows the difference seen in the IMB MPI1 Allreduce test when using MPT 3.0 on a quadcore Cray XT system versus using MPT 2.0.

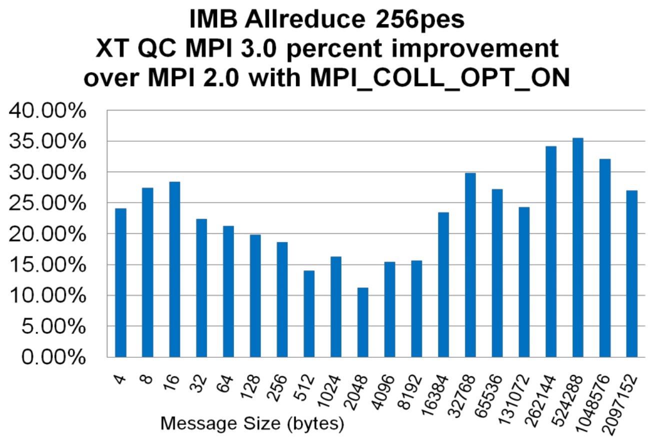

Figure 2. Improvements using SMP aware MPT 3.0

Another benefit of the MPT 3.0 implementation of the SMP device is that the performance of Portals has improved by being able to focus on inter-node communication. In summary, the SMP device will likely provide performance gains for codes in MPT 3.0.

3.0 Cray XT MPI Environment Variables

The Cray XT implementation of MPI has configurable runtime environment variables that are used to change the default behavior of the MPI library. The MPI environment variables perform various functions, such as displaying MPI related information, specifying an optimization to be used, and adjusting the size of message queues and data buffers. The default settings are set based on the best performance that can be achieved for most codes.

MPI environment variables are used to increase the performance of a code when they can provide knowledge of application behavior to the MPI library. For example, if it is known that the application uses long message sizes, the MPI environment variables can be used to specify specific optimizations that work well with long messages.

MPI environment variables also allow the user to decide between performance versus memory size tradeoffs. The performance of some codes may be improved by allocating more space for internal MPI buffers; however, this will result in less memory available for the application. Another type of tradeoff option for the user is to decide whether to use a flow control mechanism to help the code scale to higher PE counts, but can have a negative effect on performance. In addition, MPI environment variables give the user the flexibility of choosing message size cutoff values that are used to determine which algorithm is used for a specific MPI collective operation.

The Cray XT MPI environment variables are documented in the *intro_mpi* man page. The following sections will list the available MPI environment variables. Descriptions and examples of practical use will be provided for those environment variables that have been shown to be useful in optimizing and scaling MPI applications on the Cray XT.

3.1 General MPI Environment Variables

MPICH_VERSION_DISPLAY – rank 0 displays the version of Cray MPI being used (Default=0)
MPICH_ENV_DISPLAY – displays MPI environment variables and their values (Default=0)
MPICH_ABORT_ON_ERROR – causes program to abort and produce a core dump when an internal MPI-related error occurs (Default=0)
MPICH_CPU_YIELD – causes MPI process to call the *sched_yield* routine to relinquish the processor (Default=0, unless MPI detects oversubscription of CPUs.
PMI_EXIT_QUIET – inhibit PMI from displaying information about each rank (Default=0)

Table 1. General MPI Environment Variables

MPT 3.0 introduced new environment variables to display the version of the Cray XT MPI library being used (MPICH_VERSION_DISPLAY) and the settings of the MPI environment variables (MPICH_ENV_DISPLAY). Using the environment variable MPICH_ENV_DISPLAY has proven to be helpful when performing experiments to test the performance of various MPI environment variable settings. An example of the display of these variables is provided in Appendix A.

The MPICH_ABORT_ON_ERROR environment variable has been enhanced in MPT 3.0 to incorporate the functionality of the MPICH_DMASK=0x200 option from MPT 2.0. When this variable is enabled, a core file will be created when an internal MPI related error occurs, whether it is in the Portals code or MPICH2 code. Another important enhancement made in MPT 3.0 is that many MPI error messages contain more information on how to resolve the error. For example, if a buffer is being overflowed, the error message will output the current value of the relevant MPI environment variable and will suggest an action to resolve the problem.

A difference between Catamount and CNL is that CNL will allow the oversubscription of CPUs. Cray MPI will detect if CPUs have been oversubscribed and will have the process call *sched_yield* while in the MPI progress engine. A specific case where the MPICH_CPU_YIELD environment variable needs to be used is when CPU affinity is set. The PathScale compiler on OpenMP codes will set CPU affinity if the number of OpenMP threads is set to more than one. A user will need to enable MPICH_CPU_YIELD in this case.

A new feature of MPT 3.0 is the use of the Process Manager Interface (PMI) to launch MPI processes on the node. A PMI daemon is started on each node. If a MPI process aborts, it will sends a signal to the PMI daemon and the daemon will print out a message. For jobs running on many nodes, the user may want to enable PMI_QUIET_EXIT to suppress these messages. The option will not prevent Cray MPI errors from being displayed.

3.2 Rank Reorder MPI Environment Variables

MPICH_RANK_REORDER_DISPLAY – displays the node where each MPI rank is executing (Default=0)
MPICH_RANK_REORDER_METHOD – sets the MPI rank placement scheme (Default determined by job launcher, *yod* or *aprun*) **0** -> Round robin placement **1** -> SMP-style placement **2** -> Folded rank placement **3** -> Custom rank placement

Table 2. Rank Reorder MPI Environment Variables

The rank reorder feature allows the user to specify how PE ranks should be distributed on the compute nodes. Appendix B shows an example of each of the four rank reorder methods and how the ranks are distributed across 8 dual core nodes. The new MPT 3.0 environment variable MPICH_RANK_REORDER_DISPLAY is used to enable the display of the rank placement. The default rank placement is determined by the job launcher. For Catamount, *yod* will default to the round-robin placement, while CNL *aprun* has SMP-style placement as its default.

Codes have seen significant performance gains from switching from one rank reorder method to another. In general, programs tend to perform better using the SMP-style method. The reason for this is that SMP-style placement usually maps well with codes communicating with their nearest neighbor. As a result, users of Cray XT Catamount systems will want to try the SMP-style reorder method on their codes.

The use of rank reorder can be very helpful in performing load balancing. Figure 3 shows a graph of CPU utilization for an MPI job. The graph shows higher ranks having lower CPU utilization. For multi-core systems, this type of program can benefit from using the folded-rank reorder method. Using the folded-rank method would match ranks that have higher CPU utilization with ranks that have lower CPU utilization. The overall effect would improve the load balancing of the nodes.

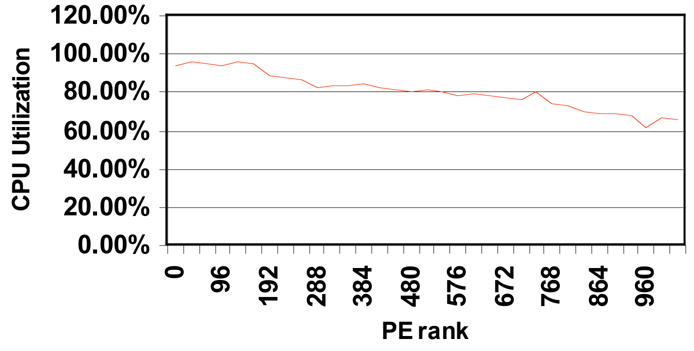

Figure 3. Rank Reorder Load Balancing Example

The use of the custom rank reorder method can provide performance improvement in cases where the data decomposition of the problem is known. In figure 4, the left side shows a SMP-style placement where four cores on a quadcore node have three links that perform intra-node communication. The right side shows how custom rank placement can be used to choose the placement scheme so that the four ranks have four links of intra-node communication. Using custom rank reorder placement to increase the number of ranks communicating on-node, will likely result in performance gains.

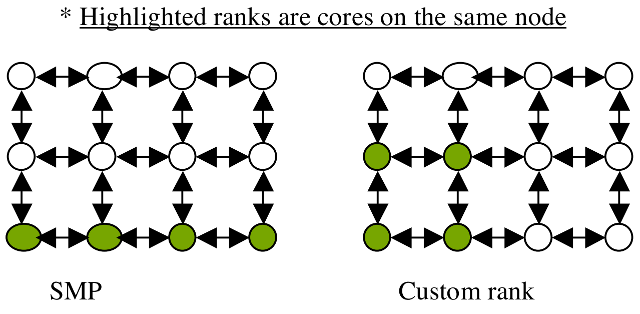

Figure 4. Custom rank reorder example

Another example of where rank reorder has shown performance improvements is in I/O. In figure 5, assume that four ranks of a 64 rank job are responsible for writing to a specific file. If the four ranks reside on different nodes, there will be four separate write requests to the file. If the four ranks reside on the same node, the four write requests from the ranks will be buffered by the Linux system and result in one write request to the file.

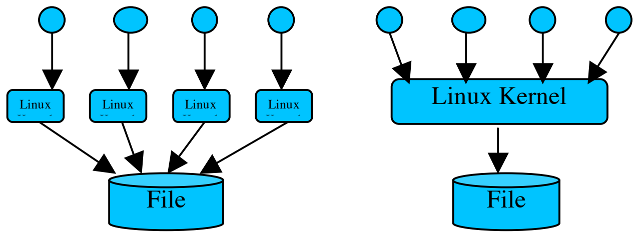

Figure 5. I/O rank reorder example

The optimal rank reorder method is not always obvious, as there may be competing performance issues of communication distance versus load balancing. Cray MPI allows the user to easily switch from one rank reorder method to another, so it is worthwhile to experiment with the different methods.

3.3 SMP MPI Environment Variables

MPICH_SMP_OFF – turns off SMP device (Default=0)
MPICH_MSGS_PER_PROCS – maximum number of internal MPI message headers (Default=16384)
MPICH_SMPDEV_BUFS_PER_PROCS – number of buffers allocated for the SMP device (Default=32)
MPICH_SMP_SINGLE_COPY_SIZE – minimum buffer size to use single-copy transfers for on-node messages (Default=128 kilobytes)
MPICH_SMP_SINGLE_COPY_OFF – disables single-copy mode (Default=0)

Table 3. SMP MPI Environment Variables

As discussed in section 2.2, the introduction of the SMP abstract device interface in MPT 3.0 will likely result in codes running faster using MPT 3.0 than using MPT 2.0. The current experience of MPT 3.0 has shown that most codes have not needed to adjust the SMP related MPI environment variables to improve performance or increase scaling. One rare case where using MPICH_SMP_OFF showed improvement was a benchmark code that pre-posted receives and benefited from using Portals matching. Another rare case where a SMP MPI variable needed to be used was that of environment variable MPICH_MSGS_PER_PROCS needing to be increased when scaling an application to a very high number of PEs.

The variable MPICH_SMP_SINGLE_COPY_SIZE is used to set the minimum size to be used for SMP single-copy transfers. The default value of 128K bytes was determined through testing. For message sizes less than 128K, it was found that the system overhead of a single-copy exceeded the time to perform the extra copy.

3.4 Portals MPI Environment Variables

The MPI Portals environment variables are used to provide information to Portals on the nature of the application's usage of MPI communication. One specific piece of information that assists in improving performance of Portals is the size of the messages that are to be used. As discussed in section 2.1, the *short message eager* protocol is faster than the *receiver-pulled long message* protocol. If an application's message size is known to be larger than the default short message size of 128,000 bytes, the environment variable MPICH_MAX_SHORT_MSG_SIZE can be increased to have these larger size message use the *short message eager* protocol. The consequence of using this variable is that the size of the Portals *unexpected buffer* may need to be increased by setting the environment variable MPICH_UNEX_BUFFER_SIZE, which will result in more memory being used for Portals communication. Conversely, if an application is not able to increase the *unexpected buffer* size large enough to allow an application to execute, then the MPICH_MAX_SHORT_MSG_SIZE may need to be decreased. A smaller value will decrease the number of messages that use the *short message eager* protocol and reduce the size of the *unexpected buffer* that is needed.

MPICH_MAX_SHORT_MSG_SIZE – maximum size of a message to be sent using *eager short message* protocol (Default=128000 bytes)
MPICH_UNEX_BUFFER_SIZE – size of the Portals unexpected message buffer (Default=60 megabytes)
MPICH_PTL_UNEX_EVENTS – number of event queue entries for unexpected messages (Default = 20480)
MPICH_PTL_OTHER_EVENTS – number of event queue entries for messages other than unexpected messages (Default=2048)
MPICH_VSHORT_OFF – disables vshort path optimization (Default=0)
MPICH_MAX_VSHORT_MSG_SIZE – maximum message size to be used for Vshort messages (Default=1024)
MPICH_VSHORT_BUFFERS – number of vshort buffers (Default=32)
MPICH_PTL_EAGER_LONG – enables *eager long message protocol* (Default=0)
MPICH_PTL_MATCH_OFF – disables registration of receive requests with portals (Default=0)
MPICH_PTL_SEND_CREDITS – enables flow control to prevent the Portals event queue from being overflowed (Default=0)

Table 4. Portals MPI Environment Variables

The *long message eager* protocol is rarely used for codes, because the application must insure that it pre-post receives in order to benefit performance. For those applications that always pre-post receives, then turning on MPICH_PTL_EAGER_LONG will improve performance. The IMB Sendrecv benchmark is an example of a code that benefits by using the *eager long message* protocol.

Applications that frequently use messages of size greater than 1K and less than 16K may benefit by making these *vshort* messages. This optimization is enabled by setting the MPICH_MAX_VSHORT_MSG_SIZE environment variable accordingly.

Latency sensitive applications may be able to improve performance by setting the environment variable MPICH_PTL_MATCH_OFF. Setting this variable disables the registration of receive requests with Portals, which eliminates a Portals system call and the Portals lock that occurs during the system call. The receive requests will be handled by the MPI layer, which will result in an extra copy of the data. The extra copy adds significant overhead for larger message sizes, so disabling portals receive matching is only beneficial for message sizes of 4K or smaller.

The Portals MPI environment variables may need to be modified as the program grows in scale. The default number of entries in the *unexpected event queue* is 20480. Since two entries are needed for each receive, the number of *unexpected event queue* entries will likely need to be increased when using 10,000 or more PEs. In addition, there are several MPI applications that exhaust the *unexpected event que*ue even at smaller number of PEs. The user is able to increase the number of *unexpected event queue* entries by modifying the MPICH_PTL_UNEX_EVENTS environment variable. The increase of the *unexpected event queue* may require a corresponding increase of the *unexpected buffer*, which is achieved by modifying the environment variable MPICH_UNEX_BUFFER_SIZE.

Some MPI applications have cases where the number of sends run well ahead of the number of receives, and the size of the *unexpected receive queue* can not be increased enough to handle this situation. In this case, a flow control mechanism can be enabled to prevent the *unexpected event queue* from being exhausted. The MPICH_PTL_SEND_CREDITS environment variable is used to enable this flow control mechanism. Setting this variable to '-1' should prevent the *unexpected receive queue* from overflowing in any situation

The *other event queue* handles events such as the sending of data, pre-posted receives, and RMA requests. Like the *unexpected receive queue*, the number of entries in the *other event queue* can also be exhausted. The overflow of the *other event queue* generally occurs less frequently than issues with the *unexpected event queue*. The MPICH_PTL_OTHER_EVENT environment variable is used to modify the size of the *other event queue*. It should be noted that in MPT 2.0, the total number of entries in the *unexpected event queue* and *other event queue* were limited to approximately 95000 entries. This restriction no longer exists in MPT 3.0.

The Cray XT system at Oak Ridge National Laboratory has run MPI test programs using 30,000 PEs. The following steps were taken to execute the IMB MPI_Alltoall program on 30,000 PEs:

- Total memory that is required just for MPI_alltoall communication: 2 * 30000 * <message-size>
  Used a maximum message size of 16K, as this will use almost 1GB of memory (running on a machine with 2GB per CPU)

- The number of *unexpected event queue* entries should be at least 2 times the number of PEs. Environment variable **MPICH_PTL_UNEX_EVENTS** was set to 70000.

- The size of the Portals *unexpected buffer* needed to be increased. Setting the MPI environment variable **MPICH_UNEX_BUFFER_SIZE**=100M was found to be sufficient

As mentioned in section 3.1, many of the Cray MPI error messages have been improved in MPT 3.0 to show more information on how to resolve the error. Table 5 shows some example error messages that may occur when using an increased number of PEs to execute the program. In the examples, the preceding number inside the brackets is the rank number that reported the error.

[12] 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).
[241] MPICH has run out of unexpected buffer space.Try increasing the value of env var MPICH_UNEX_BUFFER_SIZE (cur value is 62914560), and/or reducing the size of MPICH_MAX_SHORT_MSG_SIZE (cur value is 128000).
[233] 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).

Table 5. Sample MPT 3.0 Cray XT MPICH errors

3.5 MPI Collectives Environment Variables

One significant feature of MPT 3.0 is that the collective optimizations that were available in MPT 2.0 by enabling the MPICH_COLL_OPT_ON environment variable are now enabled by default in MPT 3.0. The new MPT 3.0 MPICH_COLL_OPT_OFF environment variable is now available to turn off collective optimizations, though there should not be reason to use this option.

The MPICH_FAST_MEMCPY environment variable is a frequently used option to increase performance of programs that use large message sizes. This feature instructs the Cray MPI routines to use a different version of memcpy that works well with message sizes of 256 kilobytes or larger. For buffer sizes less than 256K, the fast memcpy will work as well as the default memcpy. The reason this optimization in not enabled by default is that other factors, such as caching effects, may result in some applications and benchmarks performing worse when fast memcpy is used. In general, the MPICH_FAST_MEMCPY environment variable should be used when message sizes are greater than 256K.

In MPT 3.0, the default value for the environment variable MPICH_ALLTOALL_SHORT_MSG was changed from 512 to 1024. Experience has shown that programs will greatly benefit if they can use the Alltoall store and forward algorithm available for smaller message sizes. The default MPT 3.0 values of the environment variables relating to the MPI_Alltoallv and MPI_Alltoallw collectives were also modified. Testing showed using smaller values to enable flow control reduced the amount of network contention and resulted in an overall increase in performance.

MPICH_FAST_MEMCPY – use optimized memcpy routine (Default=0)
MPICH_COLL_OPT_OFF – disable collective optimizations (Default=0)
MPICH_BCAST_ONLY_TREE – set to ‘0’ to enable the MPI_Bcast ring algorithm for MPI communicators of nonpower of two (Default = 1)
MPICH_ALLTOALL_SHORT_MSG – cutoff point for using the store and forward Alltoall algorithm (Default=1024 bytes)
MPICH_REDUCE_SHORT_MSG – cutoff point for using a binomial tree algorithm (smaller values) versus a reduce-scatter algorithm (Default=65536 bytes)
MPICH_ALLRFEDUCE_LARGE_MSG – cutoff point for the SMP aware MPI_allreduce algorithm (Default=262144 bytes)
MPICH_ALLTOALLVW_FCSIZE – cutoff size of the number of communicators when the flow control (“FC”) MPI_Alltoall[vw] algorithm is used (Default=32)
MPICH_ALLTOALLVW_RECVWIN – the number of concurrent Irecv operations allowed when the flow control MPI_Alltoall[vw] algorithm is used (Default=20)
MPICH_ALLTOALLVW_SENDWIN – the number of concurrent Isend operations allowed when the flow control MPI_Alltoall[vw] algorithm is used (Default=20)

Table 6. MPI Collectives Environment Variables

In MPT 3.0, the MPI collectives have been enhanced to become SMP aware. An example of this is the MPI_Allreduce function, which was changed in MPT 3.0 to use a SMP-aware algorithm. The SMP-aware algorithm performs better than the MPICH2 algorithm for messages of size 256K or less. The environment variable MPICH_ALLREDUCE_LARGE_MSG was introduced to allow the user to adjust this cutoff point. It is expected that future versions of MPT will have additional SMP aware collective optimizations.

3.6 MPI-IO Hints Environment Variables

MPICH_MPIIO_HINTS_DISPLAY – rank 0 displays the names and values of the MPI-IO hints (Default=0)
MPICH_MPIIO_HINTS – sets the MPIO-IO hints for files opened with the MPI_File_Open routine

Table 7. MPI-IO Hints Environment Variables

MPI-IO hints are used to provide information to a MPI program to assist the performance of the MPI file I/O routines. A MPI-IO hint is specified in a MPI code in the following manner:

1. MPI_Info_create used to create an Info Object
2. MPI_Info_set adds hints to the Info Object
3. MPI_File_open passes the Info Object argument

A new feature of MPT 3.0 is the introduction of the MPICH_MPIIO_HINTS environment variable that passes the specified MPI-IO hints when the program calls MPI_File_open. The MPI-IO hints specified by MPICH_MPIIO_HINTS will override the default MPI-IO hints or any hints originating in the program by a call to the MPI_Info_set routine. Using this environment variable allows the user to test the effect on performance of various MPI-IO hints without having to modify the code or rebuild the program. The environment variable MPICH_MPIIO_HINTS_DISPLAY is helpful in displaying the hints that are used during testing. Once optimal hints are determined, they can be added to the program's code. Table 8 lists the MPI-IO hints that are supported on Cray XT systems.

direct_io	cb_nodes	romio_ds_write
romio_cb_read	cb_config_list	ind_rd_buffer_size
romio_cb_write	romio_no_indep_rw	ind_wr_buffer_size
cb_buffer_size	romio_ds_read	

Table 8. Cray XT MPI-IO hints

The MPIO-IO hint *direct_io* in MPT 3.0 has been modified in MPT 3.0, so that it verifies that data is page-aligned. If the data is page-aligned, then direct I/O will be used; otherwise the I/O will be performed without using direct I/O. Figure 6 shows a comparison of a program running with and without the MPI-IO hint *direct_io*. Note that direct I/O did not always perform better than regular buffered I/O, however, the direct I/O results were consistent between runs. The reason for this is that buffered I/O performance is more affected by the availability of the Linux kernel to perform the I/O. Another example where users have used MPI-IO hints to improve I/O performance of specific MPI codes is the enabling of collective buffering on writes (*romio_cb_write*). Also, disabling data sieving on writes (*romio_ds_write*) has been used effectively to increase I/O performance of a MPI program.

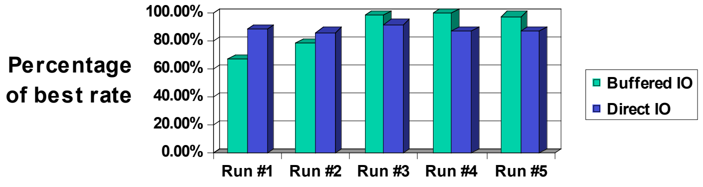

Figure 6. Direct I/O and Buffered I/O comparison

3.6 RMA MPI Environment Variables

The Cray XT implementation supports MPI-2 remote memory access (RMA) functionality. In Portals, RMA uses a message buffer that is separate from other MPI communication. The MPICH_RMA_BUFFER_SIZE environment variable is used to modify the size of the message space for RMA messages. In order to resolve flow control issues that may occur when using remote memory access communication, the environment variable MPICH_RMA_MAX_OUTSTANDING_REQS can be used. Decreasing the value of this variable will allow programs that encounter RMA overflow problems to execute. This variable has only known to be used for RMA benchmark test codes.

MPICH_RMA_BUFFER_SIZE – size of the Portals RMA message buffer (Default=6M)
MPICH_RMA_MAX_OUTSTANDING_REQS – maximum number of outstanding RMA requests a process may have for a given window (Default=64)

Table 9. RMA MPI Environment Variables

4.0 Summary

The following lists briefly summarize the Cray MPI environment variables that have been found to be useful to optimize the performance of a code or to increase the scale of the code.

MPI Environment Variables to Optimize Performance

- **Rank Reorder** – Easy to switch rank reordering methods, may result in performance improvement
    - Codes on Cray XT Catamount systems should at least try SMP-style rank placement
- **Latency sensitive codes** – Set environment variable MPICH_PTL MATCH_OFF
    - Message sizes should be 4K or less
- **Message sizes < 16K** – Setting environment variable MPICH_MAX_VSHORT_SIZE may increase performance
- **Message sizes > 128000** – Increasing environment variable MPICH_MAX_SHORT_SIZE will allow more messages to use the faster *eager short message* protocol
- **Message size > 256K** – Enable use of the fast memcpy routine by setting MPICH_FAST_MEMCPY
- **Using MPT 2.0** – Setting MPICH_COLL_OPT_ON will turn on collective optimizations that are on by default in MPT 3.0

MPI Env. Variables to Increase the Scale of the Code

- **MPICH_PTL_UNEX_EVENTS** - the number of Portals *unexpected event queue* entries should be at least two times the number of PEs being used.
- **MPICH_UNEX_BUFFER_SIZE** – the size of the Portals *unexpected buffer* may need to be increased if the number of Portals *unexpected event queue* entries is increased (MPICH_PTL_UNEX_EVENTS), or if the size of messages using the *short eager* protocol is increased (MPICH_MAX_VSHORT_SIZE).
- **MPICH_MAX_SHORT_SIZE** – If the *unexpected buffer* (MPICH_UNEX_BUFFER_SIZE) can not be increased enough, then the size of the messages eligible for the *short message eager* protocol may need to be made smaller to reduce the amount of messages being copied to the *unexpected buffer*.
- **MPICH_PTL_SEND_CREDITS** – If Portals *unexpected event queue* can not be increased high enough then a flow control mechanism needs to be enabled. Setting MPICH_PTL_SEND_CREDITS to '-1' will invoke a flow control algorithm to prevent the *unexpected event queue* from being exhausted.
- **MPICH_PTL_OTHER_EVENTS** – The number of Portals *other event queue* entries may need to be increased as the program is scaled to higher number of PEs.
- **MPICH_MSGS_PER_PROCS** – The number of internal MPI messages may possibly need to be increased at very high PE counts.

5.0 Conclusion

While the default settings for the Cray XT MPI environment variables are based to provide the best performance for most codes, there are situations where the performance and scalability of a MPI application can be improved by the use of MPI environment variables. Examples of situations where setting MPI environment variables can benefit a code have been provided. Also, explanations were given of the environment variables that may need to be modified to increase a code's scalability. Additional MPI environment variables and refinements to existing ones will be added, as Cray Inc. continues to develop new optimizations and new functionality to the Cray XT implementation of MPI

Acknowledgments

The author thanks the Cray Performance team and the Cray MPT software development group for valuable technical information and consultation.

About the Author

Geir Johansen works in Software Product Support, Cray Inc. He is responsible for support of C, C++, libc, MPI, SHMEM, TotalView and other debuggers, and performance tools for the Cray X1, Cray XT, Cray X2, and Cray XMT platforms. He can be reached at Cray Inc., 1340 Mendota Heights Road, Mendota Heights, MN 55120, USA; Email: geir@cray.com

References

[1] R. Brightwell, A. B. Maccabe, and R. Riesen. Design, Implementation, and Performance of MPI on Portals 3.0. In International Journal of High Performance Computing Applications. Vol. 17, No. 1 (2003).

[2] H. Pritchard, D. Gilmore, M. ten Bruggencate, D. Knaak, and M. Pagel. Cray Message Passing Toolkit (MPT) Software on XT3. In Cray User Group 2006 Proceedings.

[3] M. Pagel, H. Pritchard, K. McMahon, and A. Hilleary. Performance and Functional Improvements in MPT Software for the Cray XT System. In Cray User Group 2007 Proceedings.

Appendix A
MPICH_VERSION_DISPLAY and MPICH_ENV_DISPLAY Example

```
$ aprun -n 2 ./hello
MPI VERSION : CRAY MPICH2 XT version 3.0.0-pre (ANL base 1.0.4p1)
BUILD INFO  : Built Wed Mar 19  5:13:09 2008 (svn rev 6964)
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/SMP environment settings:
PE 0:   MPICH_SMP_OFF        = 0
PE 0:   MPICH_MSGS_PER_PROC        = 16384
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_BCAST_ONLY_TREE      = 1
PE 0:   MPICH_ALLTOALL_SHORT_MSG   = 1024
PE 0:   MPICH_REDUCE_SHORT_MSG     = 65536
PE 0:   MPICH_ALLREDUCE_LARGE_MESSAGE   = 262144
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          = (null)
$
```

Appendix B
MPI Rank Reorder Example

```
$ export MPICH_RANK_REORDER_DISPLAY=1
$ export MPICH_RANK_REORDER_METHOD=0
$ aprun -n 8 ./a.out
[PE_0]: MPI rank order: Using round-robin rank ordering
[PE_0]: rank 0 is on nid00469; originally was on nid00469
[PE_0]: rank 1 is on nid00470; originally was on nid00469
[PE_0]: rank 2 is on nid00471; originally was on nid00470
[PE_0]: rank 3 is on nid00473; originally was on nid00470
[PE_0]: rank 4 is on nid00469; originally was on nid00471
[PE_0]: rank 5 is on nid00470; originally was on nid00471
[PE_0]: rank 6 is on nid00471; originally was on nid00473
[PE_0]: rank 7 is on nid00473; originally was on nid00473
Application 280314 resources: utime 0, stime 0
$ export MPICH_RANK_REORDER_METHOD=1
$ aprun -n 8 ./a.out
[PE_0]: MPI rank order: Using default aprun (SMP) rank ordering
[PE_0]: rank 0 is on nid00469; originally was on nid00469
[PE_0]: rank 1 is on nid00469; originally was on nid00469
[PE_0]: rank 2 is on nid00470; originally was on nid00470
[PE_0]: rank 3 is on nid00470; originally was on nid00470
[PE_0]: rank 4 is on nid00471; originally was on nid00471
[PE_0]: rank 5 is on nid00471; originally was on nid00471
[PE_0]: rank 6 is on nid00473; originally was on nid00473
[PE_0]: rank 7 is on nid00473; originally was on nid00473
Application 280321 resources: utime 0, stime 0
$ export MPICH_RANK_REORDER_METHOD=2
$ aprun -n 8 ./a.out
[PE_0]: MPI rank order: Using folded rank ordering.
[PE_0]: rank 0 is on nid00469; originally was on nid00469
[PE_0]: rank 1 is on nid00470; originally was on nid00469
[PE_0]: rank 2 is on nid00471; originally was on nid00470
[PE_0]: rank 3 is on nid00473; originally was on nid00470
[PE_0]: rank 4 is on nid00473; originally was on nid00471
[PE_0]: rank 5 is on nid00471; originally was on nid00471
[PE_0]: rank 6 is on nid00470; originally was on nid00473
[PE_0]: rank 7 is on nid00469; originally was on nid00473
Application 280326 resources: utime 0, stime 0
```

Appendix B
MPI Rank Reorder Example

```
$ cat >MPICH_RANK_ORDER
7,2,5,0,3,6,1,4
$ export MPICH_RANK_REORDER_METHOD=3
$ aprun -n 8 ./a.out
[PE_0]: MPI rank order: Using a custom rank order from file: MPICH_RANK_ORDER
[PE_0]: rank 0 is on nid00470; originally was on nid00469
[PE_0]: rank 1 is on nid00473; originally was on nid00469
[PE_0]: rank 2 is on nid00469; originally was on nid00470
[PE_0]: rank 3 is on nid00471; originally was on nid00470
[PE_0]: rank 4 is on nid00473; originally was on nid00471
[PE_0]: rank 5 is on nid00470; originally was on nid00471
[PE_0]: rank 6 is on nid00471; originally was on nid00473
[PE_0]: rank 7 is on nid00469; originally was on nid00473
Application 280461 resources: utime 0, stime 0
$
```