Performance and Functional Improvements in MPT Software for the Cray XT System

Mark Pagel, Howard Pritchard,
Kim McMahon, Alex Hilleary
Cray Inc.

ABSTRACT: *Since the introduction of the Cray XT3, many performance improvements have been made in the MPT and portals software stack especially for dual-core support. Many of these are enabled by default but some require users to enable them. This paper will discuss these optimizations and show the measured performance improvements. In addition, key functional improvements will also be discussed. Planned MPT optimizations and functional improvements will be presented as well, including those being planned for Compute Node Linux.*

KEYWORDS: MPI, SHMEM, dual-core, portals, catamount, Compute Node Linux

1. Introduction

The software stack for Message Passing Toolkit (MPT) which contain the MPI and SHMEM libraries has been enhanced over the last year both in performance and functionality. Some of these improvements were implemented in the MPI and SHMEM libraries and some in the underlying software such as portals. This paper will discuss some of these improvements and present some measured improvements using the IMB/Pallas benchmarks. In addition, several features that should become available in the next year will also be discussed.

2. Portals Improvements

The original implementation for dual-core support was supplied by Sandia National Lab and is known as Catamount Virtual Node or CVN. This implementation used a master and slave processor paradigm for dual-core support. Although initial performance was reasonable, for some applications changes were needed at the portals layer to allow better performance. Three of these optimizations were *send-to-self short circuit optimization*, *symmetric master/slave optimization* and *portals API extensions.*

Send-to-self short circuit optimization

This optimization uses a memory to memory copy for most messages of reasonable size when the source and destination are on the same node. This reduces the latency by eliminating the round trip of the data to the SeaStar, and then back into memory. For very large messages, the data still travels out to SeaStar and back to allow additional overlap of computation. In this case, the DMA engine in the SeaStar is essentially substituting for the Opteron based memcpy function. This change was added to the 1.4.28 and 1.5.07 XT releases.

Symmetric master/slave optimization

The original master/slave design of CVN had all portals processing confined to core 0. This resolved a variety of synchronization issues. The primary downside to it is that this interferes with computation on core 0, as well as adding latency by having all core 1 portals operations forwarded to core 0 for processing. This optimization strives to reduce the latency impact by performing most all portals processing on the core that they were requested from. This means that when the process executing on core 1 issues the syscall to do a portals PUT, the processing remains on core 1, all the way through the request to the SeaStar. A single large portals lock maintains proper synchronization. Interrupts from the SeaStar are still directed to core 0 for servicing.

This optimization is most beneficial when core 1 is doing communication, while core 0 is doing computation. When both cores are actively doing communication, much of the benefit is lost due to lock contention, as well as Opteron cache thrashing. This change was added in the 1.4.28 and 1.5.07 XT releases.

Future work includes finer grained locking under CNL to reduce lock contention.

Portals API extensions

This change extended the Portals API by three new entry points. These new functions are called PtlMEMDPost, PtlMEMDInsert, and PtlMEMDAttach. All three of these functions are essentially amalgams of already existing Portals functions. The first of these (PtlMEMDPost) essentially combines the actions of PtlMEInsert, PtlMDAttach, and PtlMDUpdate. This particular sequence is used for posting an MPI receive. The combination replaces three distinct system calls with one, thereby reducing the overall cost. This change was added in the 1.4.28 and 1.5.07 XT releases.

3. MPT improvements

Several MPT optimizations have been implemented that should improve performance on user applications. Some of these have been enabled by default others get enabled when an environment variable is set. One of the optimizations that have been enabled by default is the use of PtlMEMDPost mentioned above and was made available in the 1.4.28 and 1.5.07 XT release.

Another default optimization has been improvements to the SHMEM reductions and broadcast collectives which show improvements of more than 40% over the previous versions. This optimization is in 1.4.30 and 1.5.09 versions of the XT software.

It is the goal of the MPT development group to enable optimizations by default. Some optimizatons may degrade performance in certain cases and for that reason they are not enabled by default. Further detail of these optimizations will be given here. It is our hope that many of these will be enabled by default in future releases as they are improved to handle all performance degradations. These recently added non-default optimizations are *rank placement, multi-core optimized collectives, optimized memcpy* and *disabling of portals message matching*.

Rank Placement

The default *yod* rank placement on dual-core XT systems may not be optimal for many applications. For example, for a yod-launched 8 process (4 node) MPI job on dual-core nodes, the default placement would be:

Node	0	1	2	3
Rank	0&4	1&5	2&6	3&7

A new MPI environment variable has been implemented that allows users to change this placement. The MPICH_RANK_REORDER_METHOD environment variable can be set to three different placement schemes. Setting the variable to "1" will cause MPI ranks to be assigned according to SMP placement. For example, using the SMP method an 8 process (4 node) MPI job on dual-core nodes, the rank placement would be:

Node	0	1	2	3
Rank	0&1	2&3	4&5	6&7

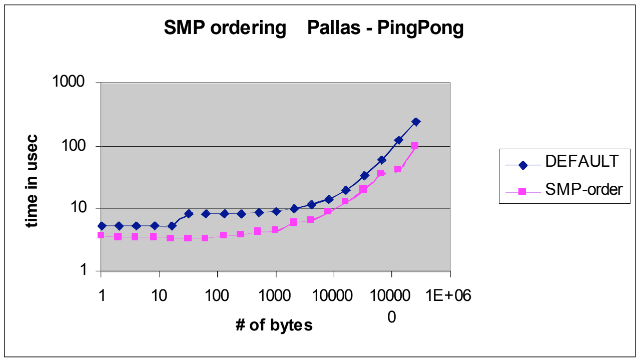
Setting MPICH_RANK_REORDER_METHOD to "2" will cause ranks to be assigned according to a folded-rank placement and has been known to help some applications. Instead of rank placement starting over on the first node when half of the MPI processes have been placed, this option places the N/2 process on the last node, going back to the initial node. For example, using the folded-rank method an 8 process (4 node) MPI job on dual-core nodes, the rank placement would be:

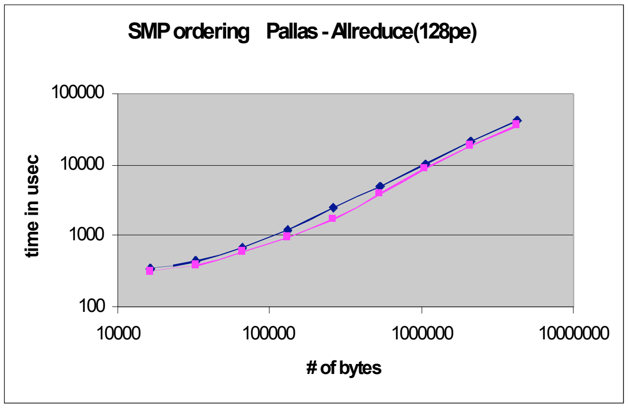
Node	0	1	2	3
Rank	0&7	1&6	2&5	3&4

Finally, some applications may benefit from fully specifying how ranks are placed and setting MPICH_RANK_REORDER_METHOD to "3" will enable custom rank placement. The custom rank placement option will cause the MPI library to look in the file called MPICH_RANK_ORDER for placement information. A comma separated list of ranks as well as hyphenated syntax is supported and can be mixed. The MPICH_RANK_ORDER file must be readable by all ranks in the current running directory. The order in which the ranks are listed in the file determines which ranks are placed closest to each other. This is most helpful for dual-core nodes. For example:

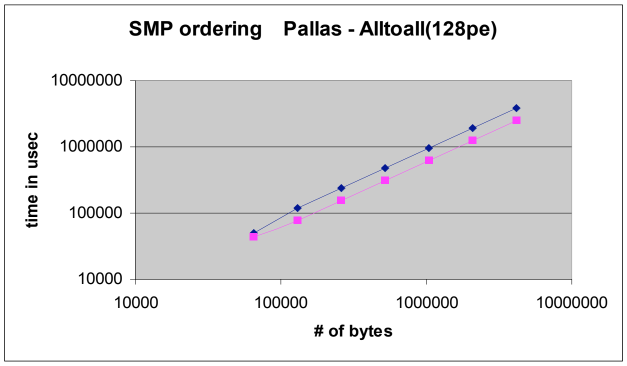
Syntax	Rank Placement
0-15	Places the ranks in SMP-style order
15-0	Places ranks 15&14 on the first node, 13&12 on the next, etc.
0,5,1,4,2,3,6,7	Places ranks 0&5 on the first node, 1&4 on the next, 2&3 together and 6&7 together.

The rank placement feature was made available in the 1.4.30 and 1.5.08 XT releases. The following charts show how SMP style placement improves performance while comparing the IMB/Pallas Benchmarks for PingPong, Allreduce, Alltoall and Bcast. In the following charts the top(dark line) is the default and the lighter is with the optimization enabled. All runs were done using 128pes. The PingPong test really is comparing on-node vs. across node since we are only using the 2 pe case which with SMP ordering, rank 0 and 1 are on the same node and with default placement they are not.

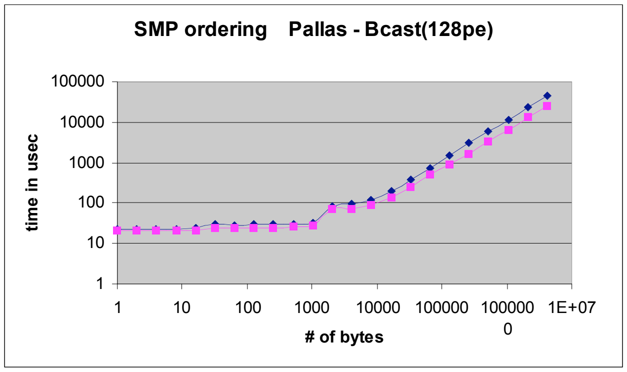




For Allreduce SMP placement was 7% to 32% faster above 16K bytes.



For Alltoall SMP placement was 13% to 36% faster above 16K bytes.



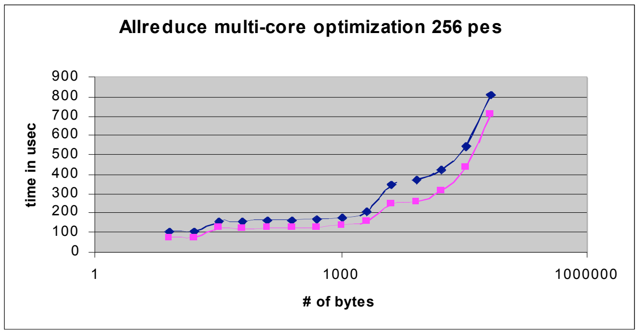
For Bcast SMP placement was 12% faster at 8 bytes and 45% faster at 1MB.

Multi-core Optimized Collectives

The XT MPICH2 supports a framework for implementing optimized collective functions, leveraging infrastructure already in place within the standard MPICH2 distribution. The framework is structured to take advantage of multi-core nodes: a network based component utilizes portals, while an on-node component can take advantage of any possible on-node communication mechanism (shared memory, for example).

Currently in the XT MPICH2 running on catamount compute nodes, portals is used both for intra-node and inter-node communication.

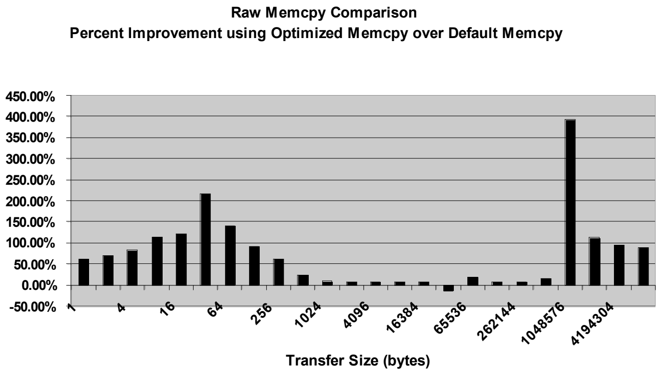
At this time only two collective operations are actually defined in the framework: MPI_Allreduce and MPI_Barrier. These optimizations can be enabled by setting the MPI_COLL_OPT_ON environment variable. The following chart shows the speed-up of MPI_Allreduce using the IMB/Pallas benchmarks. The range of speed-up was around 5% at 1MB to 40% at 8 bytes over the default for MPI_Allreduce. MPI_Barrier was about 30% faster up to 256 pes than the default.



Other functions, either implemented internally in the MPICH2 library, or in external libraries, can readily be incorporated into the framework. This optimization was introduced in the 1.4.32 and 1.5.11 XT releases.

Optimized Memcpy

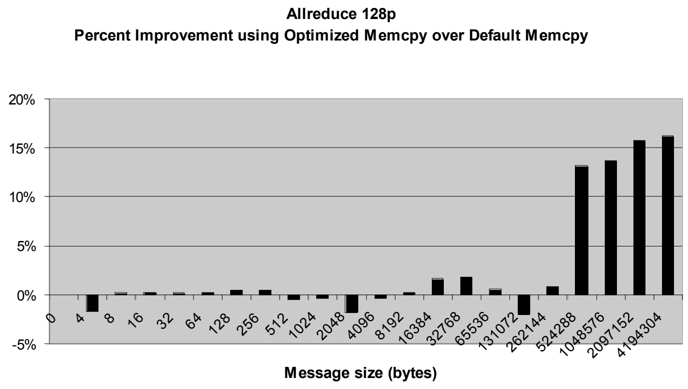
Setting the MPICH_FAST_MEMCPY environment variable enables use of an optimized memcpy routine in the XT MPICH2. This optimized memcpy routine, written in assembly for the XT architechture makes use of such features as loop unrolling, non-temporal memory references and prefetching to provide superior memcpy performance. When compared to the default Catamount memcpy performance, the optimized memcpy is nearly 4X faster for 1 megabyte copies. As noted in the chart below, the memcpy performance varies greatly with the size of the data being copied.



In MPICH, when the optimized memcpy is enabled, it is used for all local memory copies in the point-to-point and collective MPI operations. This includes copying local data out of the unexpected buffers into the user buffers on the receive-side, copying user data to internal MPI buffers on the send-side, and creating temporary copies of data buffers for some collective operations. The optimized memcpy is not currently used in the MPI I/O routines, or for derived data types.

Given the different characteristics of the memcpy routines, including cache effects, the effect of using the optimized memcpy in MPICH is not as straight forward as the raw memcpy performance indicates. Even though the optimized memcpy is faster in raw performance, some MPI collectives, as seen with the Pallas benchmark suite, show a performance degradation with certain message sizes when using the optimized memcpy.

Other message sizes show a substantial performance gain. For this reason, we have chosen not to enable the optimized memcpy as the default, but to allow users to select this optimized memcpy if their application performance can benefit from it.



For Allreduce the fast memcpy shows more than a 13% performance improvement above 256K bytes. This optimization was added to 1.4.46 and 1.5.30 XT releases.

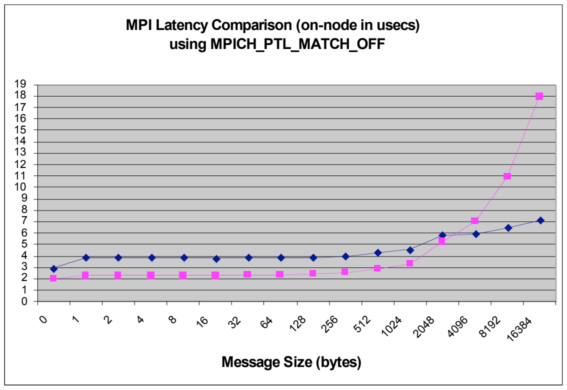
Disabling of Portals Message Matching

Setting the MPICH_PTL_MATCH_OFF environment variable disables registration of receive requests with portals. This option can significantly improve performance for latency-sensitive applications, especially on dual-core nodes. When setting this variable, receive requests (via MPI_Recv() and it's variants) are never posted to the portals-managed queue, but instead are tracked and matched via an internal MPI queue. When the application posts a receive request, setting MPICH_PTL_MATCH_OFF saves the cost of the portals system call, as well as eliminating the portals lock that is held during that portals call. When a message is received via portals, since no receive requests are posted to portals, all messages are copied to the unexpected message buffers. MPI polls the portals event queues, and when the message transfer to the unexpected buffer is complete, MPI copies the data to the user buffer specified in the receive request. An extra copy is introduced is this path, but for small mesages (less than 4096 bytes) that time is mimimal.

When messages become larger (4096 bytes - 128K bytes), the extra local copy can become significant, and in some cases, disabling the portals receive matching causes a performance degradation in MPI messaging.

For very large messages sent via the rendezvous protocol (greater than 128K bytes), MPI uses the portals 'GET' protocol. In either case, whether MPI or portals is performing the matching, no extra copy of the data is incurred, since portals only initially sends a message header. Once this header is matched to a posted receive (either via portals or MPI matching), then the GET is invoked, and the data is transfers directly into the user specified buffer. Performance for these very large message transfers usually does not change if MPICH_PTL_MATCH_OFF is set or not.

Due to the nature of how portals performs a GET, subtle timing differences in the initiating of these requests, especially when performing bi-directional large data transfers on dual-core nodes, can play a significant role in the performance. Certain sequencing of events can lead to the serialization of bi-directional traffic in Portals. We have seen cases where setting MPICH_PTL_MATCH_OFF can almost double bi-directional large message bandwidth, as well as cases where this variable has little or no effect on bandwidth. The chart below shows the improvement in PingPong latency with small message sizes when MPICH_PTL_MATCH_OFF is set and how it gets worse for larger messages.



One could imagine a superior performing solution would be to have MPI handle the message matching for small messages, and portals handle the matching for larger messages. However, this requires careful analysis, as it can be difficult to preserve the MPI ordering semantics in this scenario, especially with the posting of ANY_SRC receives. The MPT team will be looking into providing this functionality in the future. This optimization was enabled in the 1.4.50 and 1.5.39 XT releases.

4. MPT Functional Improvements

In addition to the previously mentioned performance improvements several functional improvements have been recently added. These include an updated intro_mpi man page, default64 support, Pathscale and GNU 4.1.1 compiler support.

The *intro_mpi* man page has been updated to contain more information about the environment variables that are supported and their default values. This man page should be checked often to learn of XT specific optimizations.

The PGI compiler supports the “-i8” and “-r8” compiler options which modify the default integer and real Fortran size to 8 bytes. Users who have existing codes that require this can now use the “-default64” option which will automatically add the PGI command-line options and pull in special versions of the MPI and SHMEM libraries.

When the Pathscale or GNU 4.1.1 modules are loaded, corresponding MPI and SHMEM libraries will automatically be loaded.

Some early XT3 users began using the “mpicc”, “mpif77” and “mpif90” commands. Although these are available, they are not officially supported and may be removed in the future. Users should use the “cc” and “ftn” commands which are fully supported and are needed to fully benefit from much of the recently added functionality.

5. Future MPT Improvements

With the XT 2.0 release support for Compute Node Linux(CNL) will be available. This will include support for MPI and SHMEM. These libraries will use “aprun” to launch and instructions on how to launch can be obtained from the *aprun* man page. Another feature of CNL is that the default placement on CNL will be SMP placement. Since CNL allows memory mapping a future version of the MPI library will support the ability to communicate within a multi-core node using shared memory rather than calling the portals interface.

The XT MPI-IO implementation is based on the ROMIO implementation from Argonne National Lab(ANL). Several ROMIO optimizations are not enabled by default for collective I/O. These can be enabled by using the MPI_Info_set routine to set these hints. For example:

```
    CALL MPI_Info_create(infoh, ierror )
    CALL MPI_Info_set(infoh, 'romio_cb_write',
       'enable', ierror )
```

We are working on a mechanism to more easily enable these hints using environment variables.

The new faster memcpy mentioned earlier has been limited to certain areas of the MPI library. Areas that are not currently using this fast memcpy are being examined to see if they will also benefit. These include MPI-IO and derived datatypes. In addition, other uses of the faster memcpy routine within the OS are also being examined.

Another feature that is planned to be supported is shared libraries on CNL.

6. Conclusion

The many improvements to the MPT software and underlying portals software stack have been shown to improve the IMB/Pallas benchmarks in many cases. The real test of these improvements are how they will perform on real applications. Some initial data for real applications look promising when using these optimizations. The MPT team is committed in developing new optimizations and new functionality to make the MPT software both easy to use and highly optimized.

About the Authors

Mark Pagel is the manager of the MPT group at Cray Inc. and can be reached by email at pags@cray.com. Howard Pritchard and Kim McMahon are key members of the MPT development team and continue to make many improvements in the XT MPT software. Alex Hilleary has been working on Portals since late 2004, with an emphasis on the SeaStar firmware.