A Comparison of Three MPI Implementations for Red Storm

Ron Brightwell

Scalable Computing Systems
Sandia National Laboratories*
P.O. Box 5800
Albuquerque, NM 87185-1110
rbbrigh@sandia.gov

Abstract. Cray Red Storm is a new distributed memory massively parallel computing platform designed to scale to tens of thousands of nodes. Red Storm has a custom network designed around the Cray SeaStar network interface and router. In this paper, we present an evaluation of three different MPI implementations for Red Storm: the vendor-supported MPICH2 implementation, and two other implementations based on MPICH 1.2.6. We discuss the differences in these implementations and show how various implementation strategies impact performance and scalability.

Key words: MPI, Red Storm, XT3, Portals, Performance.

1 Introduction

Cray Red Storm is a new distributed memory massively parallel computing platform designed to scale to tens of thousands of processors. The Cray XT3 is the official product from Cray that differs slightly from the Red Storm platform that has been installed at Sandia National Laboratories in Albuquerque, New Mexico, USA. The XT3 has a three-dimensional torus network, while the Red Storm system is torus only in one direction. This limitation allows the Red Storm system to support more easily switching large sections of the machine between classified and unclassified computing. Other than this feature, the hardware and software environment of Red Storm is identical to the XT3. Henceforth, we will simply refer to Red Storm, since that is the platform on which all of our experiments were performed.

Like any other distributed memory parallel computing platform, the performance of the network and the performance of the MPI implementation are critical to the overall performance, scalability, and, ultimately, the success of the machine. For Red Storm, Cray has designed and implemented a custom network interface and router chip, called the Cray SeaStar [1], specifically to meet the demands of a large-scale distributed

* Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

memory scientific computing machine. The network performance requirements for Red Storm were ambitious when they were first proposed in 2002. The network is required to deliver 1.5 GB/s of network bandiwidth into each compute node and 2.0 GB/s of link bandwidth. The one-way MPI latency requirement between nearest neighbors is 2 μsecand is 5 μsecbetween the two furthest nodes.

In this paper, we discuss the design, implementation, and performance of three different MPI libraries for Red Storm. Each of these libraries has different features that have different performance and scalability implications. We describe these differences and show performance results from several communication micro-benchmark tests. We also compare the performance of MPI to the performance of the lowest-level communication mechanism employed by the SeaStar. Our results show that the overhead of the MPI implementation for point-to-point message passing operations is less than 0.5 μsec.

The rest of this paper is organized as follows. The following section provides details of the hardware and software environment on Red Storm. Section 3 describes the three different implementations of MPI for Red Storm. A performance comparison of the three implementations is presented in Section 4, and relevant conclusions of this work are summarized in Section 5. Section 6 outlines plans for continued research and development activities surrounding MPI on Red Storm.

2 Red Storm

The following describes the hardware and software environment of the Red Storm system that was used for our experiments. A more detailed description of Red Storm can be found in [2].

2.1 Hardware

The Red Storm system installed at Sandia is composed of 10,368 compute nodes in 108 cabinets. The network is configured in a 27 x 16 x 24 mesh topology. Each compute node contains a 2.0 GHz AMD Opteron processor with 2 GB of main memory. Each node also contains a SeaStar network interface and router chip attached via a HyperTransport (HT) link. In addition to independent send and receive DMA engines, the SeaStar has an embedded 500 MHz PowerPC 440 processor for offloading network protocol processing activities and 384 KB of local scratch memory. The physical links in the network support up to 2.5 GB/s of data payload in each direction. The interface to the Opteron uses 800 MHz HT, which provides a theoretical peak of 3.2 GB/s per direction. After protocol overhead, the link is expected to deliver a peak payload rate of 2.8 GB/s.

2.2 Software

Compute nodes in Red Storm run a third-generation lightweight kernel called Catamount. Catamount is a follow-on to the Puma/Cougar [3] lightweight kernel that ran on Sandia's ASCI Red [4] machine. Catamount has been enhanced to provide running

applications in full 64-bit mode on the Opteron and has also undergone some small changes to integrate it into Cray's system management infrastructure. Red Storm service nodes run SuSE Linux.

The software environment for the SeaStar is based on the Portals [5] network programming interface developed jointly by Sandia and the University of New Mexico. Portals provides one-sided data movement operations between disjoint processes. However, unlike most one-sided programming interfaces, the target of a remote operation is not a virtual address or memory key. Rather, the ultimate destination of an incoming message is determined solely by the receiver when the message arrives. The receiver is responsible for putting Portals objects together in a way that meets the needs of the upper-level protocol. Portals are very much like protocol building blocks that can be combined to meet a variety of needs. In the case of Red Storm, every service that uses the network does so via Portals, whether it be loading a job onto a compute node, network-based file systems, or MPI communication.

Portals is currently an active area of development for Red Storm. The current implementation of Portals for Red Storm handles much of the protocol processing on the Opteron and does not use the PowerPC to its fullest capabilities. This approach allows for a single instance of firmware for the PowerPC that supports both the physically contiguous memory model of Catamount and the non-contiguous physical memory model for Linux. When a new message arrives at the network interface, the SeaStar interrupts the host processor, which then processes the message header, traverses the Portals data structures, and programs the DMA engines on the SeaStar appropriately. The results that we present in Section 4 are using this interrupt-driven mode. The results are quite encouraging, given the cost of using interrupts. We expect an implementation that offloads all of the protocol processing to the PowerPC on the SeaStar to be available within the next few months, and it will be interesting to compare those results as well.

3 MPI Implementations

In this section we describe the three different MPI libraries that have been implemented for Portals on Red Storm.

3.1 MPICH2-0.97

The Cray supported version of MPI for Red Storm is based on MPICH2 [6]. They have created a Portals device for MPICH2 (version 0.97) that supports all of the MPI-2 functionality except for the dynamic process creation functions and the connect/accept functions. The organization of the Portals structures and protocols used for this implementation to implement the MPI point-to-point communication operations are essentially identical to those describe in [7]. However, Cray has made a few small changes to their implementation. Rather than having each non-blocking send and receive operation use a separate Portals event queue, this implementation uses only two event queues total: one for unexpected messages and one for all other types of communication events. Cray likely took this strategy to reduce the complexity of the implementation and to reduce the amount of memory needed for event queues on the SeaStar. A drawback to this

approach is that the time needed to complete an operation is no longer specific to that operation. For example, when waiting on a message to arrive for a posted receive, this implementation must handle all other events that occur while waiting for the message to arrive. This strategy assumes that there is outstanding work to be done while waiting for communication operations to complete.

3.2 MPICH-1.2.6

This implementation was developed in support of the previous version of Portals that ran on the Cplant [8] Linux cluster at Sandia. It was ported forward to the current version of Portals that runs on Red Storm. Like the MPICH2 implementation from Cray, the Portals structures and protocols are essentially identical to the those described in [7], with one exception. This implementation has an optimization for very short messages that is similar to message "copy blocks" or "bounce buffers" used in other MPI implementations [9, 10].

Previously, the short message protocol was implemented by creating a Portals memory descriptor over the region of memory to be sent and then invoking the Portals put operation to deliver this data to the destination. In order to avoid the overhead of creating a new memory descriptor each time a short message is sent, the MPI library creates a large memory descriptor during initialization. It divides this memory up into several short message buffers. When a short message is to be sent, the MPI library copies the message into one of these buffers and sends it. From the user point of view, the send is complete because the user buffer is free to modify the buffer. From a Portals point of view, the put operation may not have completed, since the events signifying completion may still be pending. The completion events for very short messages will be consumed whenever the library is blocked waiting for an operation to complete or whenever a short message is to be sent and there are no free short message buffers available. So, in addition to avoiding the overhead of creating and destroying a memory descriptor for each short message, this optimization attempts to hide the cost of consuming events associated with short messages. This approach also helps reduce the number of memory descriptors used for non-blocking short messages. The very short message optimization can be disabled via an environment variable, making it easy to measure the performance gained by using this strategy.

3.3 MPICH-1.2.6 using SHMEM

Since Portals provides one-sided operations, it can easily support the Cray SHMEM [11] programming model (provided the operating system maps static variables at identical locations in separate processes, as Catamount does). Thus, it is possible to use the MPI implementation developed for SHMEM [12] on Red Storm. This is not a complete implementation of SHMEM (which Cray plans to provide for the XT3), but rather a small subset of SHMEM interface.

This implementation has a few advantages over the others in terms of Portals resource usage. Unlike the other two implementations, the number of Portals data structures that MPI uses is fixed. The SHMEM subset library creates eight memory descriptors and one event queue. These structures are created during MPI initialization and

remain persistent throughout the life of the process. The other implementations create and destroy memory descriptors, match entries, and sometimes event queues as the process runs. This can be an advantage for applications that have large numbers of outstanding operations where the limited amount of memory on the SeaStar is a problem. In addition, the SHMEM implementation implements flow control at the user-level, so there is no way to exhaust the resources for handling unexpected messages. The other implementations allocate a fixed amount of space during initialization for unexpected message, and if this space is exceeded, the application process is terminated with a resource exhaustion error.

The main drawback of this implementation is that it performs all of the matching semantics of MPI at the user-level. The MPI library is responsible for maintaining the posted receive queue and traversing it each time a new message arrives. The other implementations take advantage of the matching semantics of Portals.

4 Micro-benchmarks and Results

In order to measure the performance of the three MPI libraries, we use two micro-benchmarks. The first is a standard ping-pong latency and bandwidth benchmark developed at Sandia. This benchmark measures the ideal case where a receive is pre-posted. We also have a version of this benchmark that measures latency and bandwidth at the Portals level so that we can measure the overhead incurred by MPI.

The second benchmark used is the NetPIPE [13] benchmark. We used the standard MPI-1 module that comes with the distribution and measured latency and bandwidth using the standard ping-pong, ping-pong with pre-posting, bi-directional ping-pong, and streaming modes. We also developed a Portals module for NetPIPE so that we could again measure the overhead of MPI in these various modes.

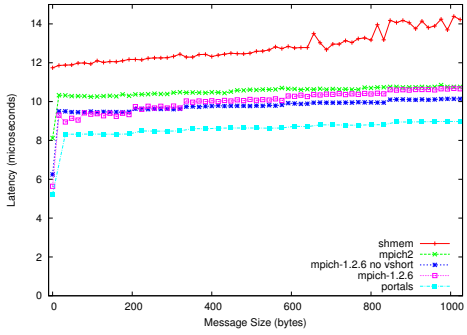

Fig. 1. MPI latency performance

Figure 1 shows the results of the Sandia benchmark for Portals, MPICH 1.2.6, MPICH 1.2.6 without the short message optimization, MPICH2, and MPICH 1.2.6 using SHMEM. The zero-length latency is 5.23 μsec, 5.64 μsec, 6.25 μsec, 8.12 μsec,

and 11.74 μsecrespectively. The MPI overhead for zero-length messages starts out relatively small, only 0.41 μsec, but eventually steadies at around 1.17 μsecfor 64 bytes and beyond. Interestingly, the very short message protocol in MPICH 1.2.6 only ends up being a win for messages smaller than 320 bytes. The very short message switch point is currently set at 8 KB, so this number will need to be tuned as Portals development continues.

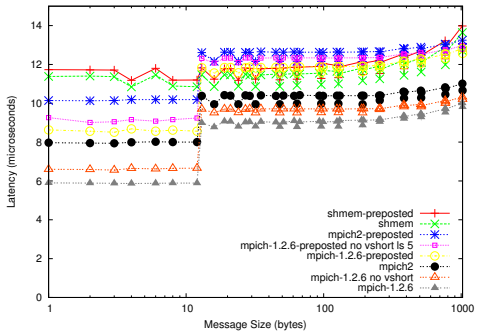

Fig. 2. NetPIPE latency performance

Figure 2 shows the NetPIPE latency performance for both the default mode of operation and for the mode that insures that a receive is pre-posted. For this test, the latency for a one-byte message for MPICH 1.2.6 is 5.9 μsec, 6.6 μsecfor MPICH 1.2.6 without the very short message optimization, and 7.97 μsecfor MPICH2. This graph clearly shows a jump at 12 bytes, which is the largest amount user data that can fit in a single Portals header packet and be serviced by a single interrupt on the SeaStar. Messages larger than 12 bytes require two interrupts to be serviced. For this test, pre-posting receives does not offer a performance gain.

Figure 3(a) shows the bandwidth results for NetPIPE. Both MPICH 1.2.6 and MPICH2 perform similarly up to the long message protocol crossover point of 128 KB. At that point, MPICH2 continues to perform well, but MPICH 1.2.6 falls off to only about 700 MB/s. The long message protocol for both implementations is eager, which means that a message will be sent across the wire twice if the message is unexpected. For this particular test, MPICH 1.2.6 gets the initial message to the receiver before the receive is posted, but since MPICH2 is slightly slower, it gets the message there after the receive has been posted. If we insure that a receive is always pre-posted, MPICH 1.2.6 performs as well as MPICH2. The asymptotic bandwidth is a little over 1.1 GB/s.

Figure 3(b) shows the bi-directional bandwidth results for NetPIPE. There is virtually no difference between MPICH 1.2.6 and MPICH2 for bi-directional bandwidth. Both are able to achieve an asymptotic bandwidth of a little more than 2.2 GB/s. Even the SHMEM implementation is able to sustain this level at very large message sizes.

Figure 4 shows the streaming bandwidth results for NetPIPE. The interesting result in this data is that the very short message optimization in MPICH 1.2.6 actually allows

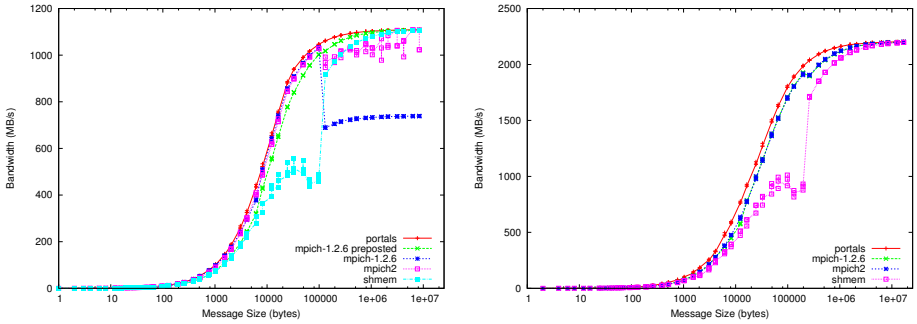

Fig. 3. NetPIPE uni-directional **(a)** and bi-directional **(b)** bandwidth

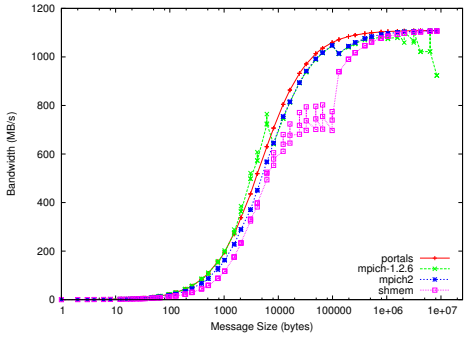

Fig. 4. NetPIPE Stream Bandwidth

for greater performance than what the raw Portals performance can provide. The Portals module in NetPIPE waits for completion events after each message is sent, while the very short message optimization does not. This overhead is reflected in messages smaller than 8 KB. Beyond 8 KB, the performance of MPICH 1.2.6 and MPICH2 are nearly identical. However, there is an unexplained drop in performance for MPICH 1.2.6 for messages greater than 1 MB. This is most likely attributed to some of the streaming messages being sent across the wire twice due to a matching receive not being pre-posted.

5 Conclusions

In this paper, we have compared the performance using micro-benchmarks of three different implementations of MPI on the Cray Red Storm platform. Despite the current interrupt-driven implementation of Portals for Red Storm, MPI zero-byte half round trip latency is a little more than 5 μsec. While this number is more than twice the requirements for Red Storm, we expect this will improve significantly once a full offload implementation of Portals is completed. The bandwidth numbers indicate that the Cray SeaStar is able to deliver more than 1.1 GB/s of uni-directional bandwidth and is able to maintain that level of performance in both directions simultaneously. The MPICH 1.2.6 version slightly outperforms the MPICH2 implementation in terms of latency and bandwidth. We have measured the overhead of MPI on top of Portals to be a little more than a microsecond for short messages and have shown that a short message optimization can also provide an increase in streaming bandwidth performance.

6 Future Work

We expect to do a much more in-depth analysis of the performance of the MPICH 1.2.6 and MPICH2 implementations using real applications. One important area of performance yet to be analyzed is collective operations. We also plan to prototype some additional features, such as a rendezvous mode to avoid sending unexpected long messages twice, in both implementations. We also plan to measure and compare the performance of the MPICH2 one-sided operations to that of the point-to-point operations.

References

1. Alverson, R.: Red Storm. In: Invited Talk, Hot Chips 15. (2003)
2. Camp, W.J., Tomkins, J.L.: Thor's hammer: The first version of the Red Storm MPP architecture. In: In Proceedings of the SC 2002 Conference on High Performance Networking and Computing, Baltimore, MD (2002)
3. Shuler, L., Jong, C., Riesen, R., van Dresser, D., Maccabe, A.B., Fisk, L.A., Stallcup, T.M.: The Puma operating system for massively parallel computers. In: Proceeding of the 1995 Intel Supercomputer User's Group Conference, Intel Supercomputer User's Group (1995)
4. Timothy G. Mattson, David Scott, S.R.W.: A TeraFLOPS Supercomputer in 1996: The ASCI TFLOP System. In: Proceedings of the 1996 International Parallel Processing Symposium. (1996)

5. Brightwell, R., Hudson, T.B., Maccabe, A.B., Riesen, R.E.: The Portals 3.0 message passing interface. Technical Report SAND99-2959, Sandia National Laboratories (1999)
6. Gropp, W.: MPICH2: A new start for MPI implementations. In Kranzlmuller, D., Kacsuk, P., Dongarra, J., Volkert, J., eds.: Recent Advances in Parallel Virtual Machine and Message Passing Interface: 9th European PVM/MPI Users' Group Meeting, Linz, Austria. Volume 2474 of Lecture Notes in Computer Science., Springer-Verlag (2002)
7. Brightwell, R., Maccabe, A.B., Riesen, R.: Design, implementation, and performance of MPI on Portals 3.0. International Journal of High Performance Computing Applications **17** (2003) 7–20
8. Brightwell, R., Fisk, L.A., Greenberg, D.S., Hudson, T.B., Levenhagen, M.J., Maccabe, A.B., Riesen, R.E.: Massively Parallel Computing Using Commodity Components. Parallel Computing **26** (2000) 243–266
9. Chaussumier, F., Desprez, F., Prylli, L.: Asynchronous Communications in MPI – the BIP/Myrinet Approach. In Dongarra, J., Luque, E., Margalef., T., eds.: Proceedings of the EuroPVM/MPI'99 conference. Number 1697 in Lecture Notes in Computer Science, Barcelona, Spain, Springer Verlag (1999) 485–492
10. Dimitrov, R., Skjellum, A.: An efficient MPI implementation for Virtual Interface (VI) Architecture-enabled cluster computing. In: Proceedings of the Third MPI Developers' and Users' Conference. (1999) 15–24
11. Cray Research, Inc.: SHMEM Technical Note for C, SG-2516 2.3. (1994)
12. Brightwell, R.: A new MPI implementation for Cray SHMEM. In: Recent Advances in Parallel Virtual Machine and Message Passing Interface: 11th European PVM/MPI Users' Group Meeting. (2004)
13. Snell, Q.O., Mikler, A., Gustafson, J.L.: NetPIPE: A network protocol independent performance evaluator. In: Proceedings of the IASTED International Conference on Intelligent Information Management and Systems. (1996)