An Evaluation of Open MPI's Matching Transport Layer on the Cray XT

Richard L. Graham[1], Ron Brightwell[2], Brian Barrett[3],
George Bosilca[4], and Jelena Pješivac-Grbović[4]

[1] Oak Ridge National Laboratory*
Oak Ridge, TN USA
rlgraham@ornl.gov

[2] Sandia National Laboratories**,
Albuquerque, NM USA
rbbrigh@sandia.gov

[3] Los Alamos National Laboratory***
Los Alamos, NM USA
bbarrett@lanl.gov

[4] The University of Tennessee,
Knoxville, TN USA
{bosilca,pjesa}@cs.utk.edu

Abstract. Open MPI was initially designed to support a wide variety of high-performance networks and network programming interfaces. Recently, Open MPI was enhanced to support networks that have full support for MPI matching semantics. Previous Open MPI efforts focused on networks that require the MPI library to manage message matching, which is sub-optimal for some networks that inherently support matching. We describes a new matching transport layer in Open MPI, present results of micro-benchmarks and several applications on the Cray XT platform, and compare performance of the new and the existing transport layers, as well as the vendor-supplied implementation of MPI.

1 Introduction

The Open MPI implementation of MPI is the result of an active international open-source collaboration between industry, research laboratories, and academia. In a short time, Open MPI has evolved into a robust, scalable, high-performance implementation for a wide variety of architectures and interconnects. It is

* Research sponsored by the Mathematical, Information, and Computational Sciences Division, Office of Advanced Scientific Computing Research, U.S. Department of Energy, under Contract No. DE-AC05-00OR22725 with UT-Battelle, LLC.

** 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.

*** Los Alamos National Laboratory is operated by Los Alamos National Security, LLC, for the National Nuclear Security Administration of the U.S. Department of Energy under contract DE-AC52-06NA25396.

F. Cappello et al. (Eds.): EuroPVM/MPI 2007, LNCS 4757, pp. 161–169, 2007.

currently being run in production on several of the largest production computing systems in the world. Much of the current effort in developing Open MPI has targeted networks and network programming interfaces that do not support MPI matching semantics. These networks depend on the MPI implementation to perform message selection inside the MPI library. As such, existing transport layers in Open MPI were designed to provide this fundamental capability. Unfortunately, these transport layers have been shown to be sub-optimal in some cases for networks that support MPI matching semantics, mostly due to redundant functionality.

Recently, a new transport layer has been developed that is designed specifically for networks that provide MPI matching semantics. This new transport layer eliminates much of the overhead of previous transport layers and exploits the capabilities of the underlying network layer to its fullest. This paper describes this new matching transport layer and its implementation on the Cray XT platform. We compare and contrast features of the new transport with the existing non-matching transport layer. Performance results from several microbenchmarks demonstrate the capabilities of the new transport layer, and we also show results from several real-world applications. We also include performance results for the native vendor-supplied MPI implementation.

The rest of this paper is organized as follows. Section 2 presents an overview of the Open MPI implementation for the Cray XT platform, the Cray MPI implementation, and the test platform for experiments presented in this paper. Results for microbenchmarks and applications are presented in Sections 3 and 4, respectively. Relevant conclusions are presented in Section 5.

2 Background

The Cray XT4 platform utilizes the Portals [1] interface for scalable, high performance communication. Portals provides a number of features not common to high performance networks, particularly rich receive matching capable of implementing the MPI message matching rules. Initial work with Open MPI on the XT4 treated Portals like traditional commodity networks [2]. Recent work extends Open MPI to take advantage of Portals' rich feature set.

2.1 Open MPI Point-to-Point Architecture

Open MPI implements point-to-point MPI semantics utilizing a component interface, the Point-To-Point Management Layer (PML) [3]. The PML is responsible for implementing all MPI point-to-point semantics, including message buffering, message matching, and scheduling message transfers. The general architecture is shown in Figure 1. At run-time, one PML component will be selected and used for all point-to-point communication. Three PMLs are currently available: OB1, DR, and CM.[1] The PMLs can be grouped into two categories based based on responsibility for data transfer and message matching: OB1 and DR or CM.

[1] PML names are internal code names and do not have any meaning.

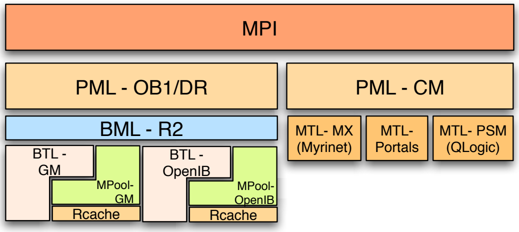


Fig. 1. Open MPI's Layered Architecture

OB1 and DR. OB1 and DR implement message matching and data transfer scheduling within the MPI layer, utilizing the BTL interface for data transfer. OB1 provides high performance communication on a variety of HPC networks and is capable of utilizing remote direct memory access (RDMA) features provided by the underlying network. DR is focused on data integrity and only utilizes send/receive semantics for message transfer. Both PMLs share the lower level Byte Transfer Layer (BTL), which is the layer that handles the data transfer, the supporting Byte Management Layer (BML), Memory Pool (MPool), and the Registration Cache (Rcache) frameworks. While these are illustrated and defined as layers, critical send/receive paths bypass the BML, which is used primarily during initialization and BTL selection.

When using OB1 and the Portals BTL, short messages are sent eagerly and long messages are sent using a rendezvous protocol. Eager message transfer involves a copy into BTL-specific buffers at the sender and a copy out of BTL-specific buffers at the receiver. For long messages, a Portals RDMA get is issued to complete data transfer directly into the application receive buffer. User-level flow control ensures messages are not dropped, even for large numbers of unexpected sends.

CM. The CM PML provides request management and handling of buffered sends, relying on the MTL framework to provide message matching and data transfer. The MTL is designed specifically for networks such as Portals or Myrinet MX, which are capable of implementing message matching inside the communication library. Unlike OB1, which supports multiple simultaneous BTLs, only one MTL may be utilized per application.

The Portals MTL utilizes a design similar to that described in [1]. The Portals MTL sends all data eagerly, directly from application buffers. If a receive has been pre-posted, the data is delivered directly to the user buffer. Unexpected short message, less than 32K currently, are buffered in MTL level buffers. Unexpected long messages are truncated, and after a match is made Portal's RDMA get functionality completes the data transfer. With the exception of unexpected receives, messages are matching by the Portals library. The Portals MTL is designed to provide optimal performance for applications that pre-post their receives.

OB1 and CM fundamentally differ in the handling of long messages. The OB1 protocol uses a rendezvous protocol with an eager limit of 32K bytes. On the receive side the memory descriptors are configured to buffer this data if messages are unexpected. For large messages, the OB1 protocol attempts to keep network congestion down, so sends only a header used for matching purposes. Once the match is made, the Portals get method is used to deliver the user's data in a zero copy mode, if the MPI data type is contiguous, directly to the destination. This mode of point-to-point communications is very useful when an application run uses a lot of large unexpected messages, i.e. when the message is sent to the destination, before the receive side has posted a matching receive.

CM does not specify a protocol for long messages, leaving such decisions to the MTL. The Portals MTL procotol is agressive on sending data. Both the short and the long protocol send all user data at once. If there is a matching receive posted, the data is delivered directly to the user destination. In the absence of such a posted receive, short messages, i.e. messages shorter than 32K bytes, are buffered by the receive Portals memory descriptor. However, all he data associated with long messages is dropped, and a Portals get request is performed after the match is made to obtain the data. This protocol is aimed at providing the highest bandwidth possible for the application.

2.2 Cray MPI

Cray MPI is derived from MPICH-2 [4], and supports the full MPI-2 standard, with the exception of MPI process spawning. This is the MPI implementation shipped with the Cray Message Passing Toolkit. The communication protocol used by Cray MPI is generally similar to that of the Portals MTL, although there are significant differences regarding the handling of event queue polling.

2.3 Application Codes

Four applications, VH-1, GTC, the Parallel Ocean Program (POP), and S3D, were used to compare the protocols available on the Cray XT platform. VH-1 [5] is a multidimensional ideal compressible hydrodynamics code. The Gyrokinetic Toroidal Code [6] (GTC) uses first-principles kinetic simulation of the electrostatic ion temperature gradient (ITG) turbulence in a reactor-scale fusion plasma to study turbulent transport in burning plasmas. POP [7] is the ocean model component of the Community Climate System Model, which is used to provide input to the Intergovernmental Panel on Climate Change assessment. S3D [8] is used for direct numerical simulations of turbulent combustion by solving the reactive Navier-Stokes equations on a rectilinear grid.

2.4 Test Platforms

Application performance results were gathered on Jaguar, a Cray XT4 system at Oak Ridge National Laboratory. Jaguar is composed of 11,508 dual-socket 2.6 GHz dual-core AMD Opterons, and the network is a 3-D torus with the Cray SeaStar 2.1 [9] network. Micro-benchmark results were gathered on Red Storm, a Cray XT3+ system at Sandia National Laboratories. Red Storm contains 13,272 single-socket dual-core 2.4 GHz AMD Opteron chips, a SeaStar

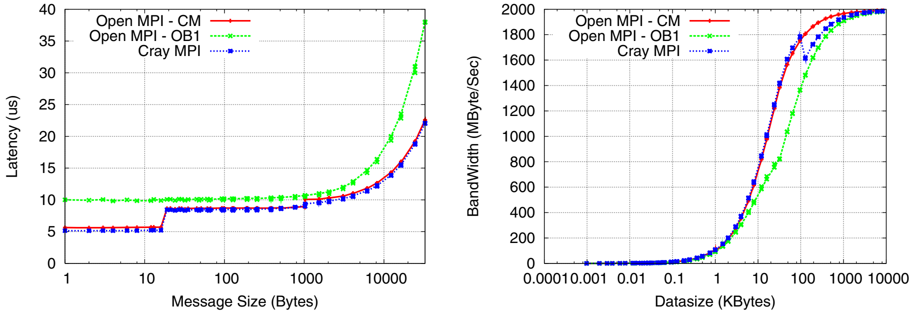
2.1 network, which is torus in only one direction. The major difference between these two systems is the speed of the processor, and the communication micro-benchmarks can be scaled appropriately. For both systems, compute nodes run the Catamount lightweight kernel, and all network communications use the Portals 3.3 programming interface [10].

For the application results, the default Cray MPI installation, XT/MPT version 1.5.31 with default settings, is used for the benchmark runs. The trunk version of Open MPI (1.3 pre-release) is used for these runs, with data collected using both the Portal ports of the CM and OB1 PMLs. Open MPI's tuned collectives are used for collective operations. To minimize differences in timings due to processor allocations, all runs for a given application and processor count are sequentially run within a single resource allocation.

3 Micro-Benchmark Performance

We use several communication micro-benchmarks to compare the performance of the two MPI implementations. We first compare latency and bandwidth performance using the NetPIPE [11] benchmark. Figure 2(a) shows half round-trip ping-pong latency results. The Cray implementation has the lowest zero-length latency at 4.78 μs, followed by 4.91 μs and 6.16 μs respectively for Open MPI's CM and OB1. Figure 2(b) plots bandwidth performance. Results shows that beyond a message length of 100 bytes, Open MPI's CM bandwidth is higher than that of Cray MPI's, but eventually the curves join as the asymptotic peak bandwidth is reached. However, Cray MPI's bandwidth curve is consistently higher than that of Open MPI's OB1 protocol. Note also that Open MPI's transition from the short-message protocol to the long-message protocol produces a much smoother curve than Cray MPI's.

Next, we measure CPU availability for sending and receiving using the Sandia overhead benchmark [12]. This benchmark measures the percentage of the processor that is available to the application process while sending and



(a) (b)

Fig. 2. NetPIPE (a) latency and (b) bandwidth performance

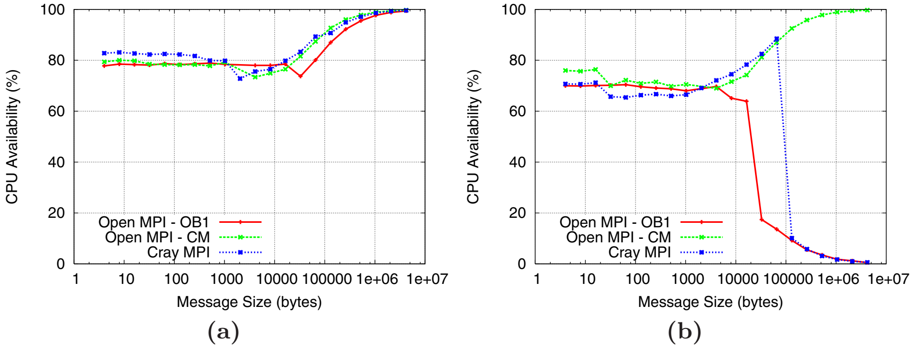


Fig. 3. SMB send availability (a) and receive availability (b)

receiving messages. Figure 3(a) shows send-side CPU availability, while Figure 3(b) shows receive-side CPU availability. On the send side, Cray MPI has a very slight advantage for very small messages sizes. However, for message sizes between 1 KB and 10 KB, the OB1 transport has a slight advantage over the other two. This is likely due to memory copies in Cray MPI and CM that reduce latency at the expense of CPU availability. Results for receive availability are much different. The CM transport has a slight advantage at small message sizes, but is able to maintain high availability for very large messages. The eager protocol messages in CM allow for nearly complete overlap of computation and communication. The other two curves show a rapid decrease in availability at the point where the eager protocol switches to a rendezvous protocol. Cray recently modified their implementation to use a rendezvous protocol by default, in spite of previous results that demonstrated high receive-side availability similar to CM.

For our last communication micro-benchmark, we examine message rate using a modified version of the Ohio State University streaming bandwidth benchmark. This benchmark measures the number of messages per second that can

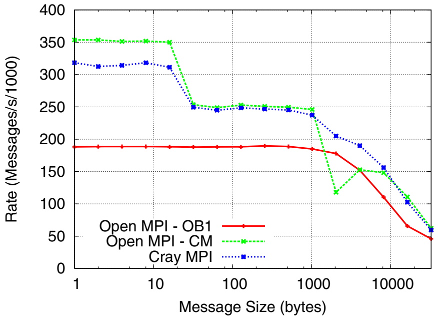


Fig. 4. Small message rate

be processed by streaming messages. In Figure 4 we can see that CM has an advantage over Cray MPI for message sizes up to about 32 bytes, at which point the curves almost converge. Performance of the CM transport drops significantly at 2048 bytes. The OB1 message rate is nearly half of the other implementations, due to both protocol overhead and rate limiting to ensure message reliability.

4 Application Performance

We compare the performance of VH-1, GTC, POP, and S3D, at medium process count. Figure 5 shows overall application run-time for these codes, with data for VH-1 and POP collected at 256 processes, and for GTC and S3D at 1024 processes. Overall, Open MPI CM PML slightly out-performs Cray MPI, and the CM PML consistently outperforming the OB1 PML.

VH-1 was run using 256 MPI processes, with the CM run-times being about 0.4% faster than the Cray MPI run-times, and about 0.3% faster than the OB1 runs. For the GTC runs at 1024 processes, the Cray MPI application run-times are about 4% faster than the Open MPI CM runs, and 15% than the OB1 runs. Running POP at 256 processes, Open MPI CM outperforms Cray MPI by about 3%, and outperforms CM by 18%. Finally, at 1024 processes, Open MPI's CM outperforms Cray MPI by 12%, and it outperforms OB1 by 3%.

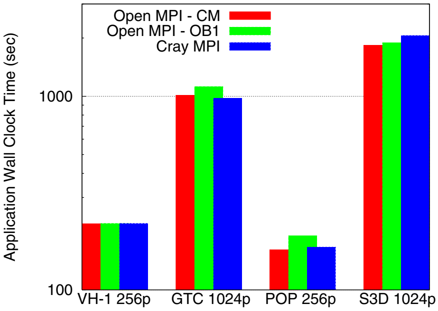


Fig. 5. Application Wall Clock Run-Time(sec)

Table 1 lists the fraction of run-time spent inside the MPI library, along with the most time consuming MPI functions. The data was collected using mpiP [13], with the CM PML. The average amount of time spent in MPI routines differs considerably from application to application, with 6.1% of GTC's run time at 1024 processes being spent in the MPI library, to 65.7% of POP's run-time at being spent in the MPI library. 7.9% of S3D's run-time and 16.9% of VH-1's run-time are spent in the MPI library. For applications other than S3D—which uses collectives sparingly—collective communications dominate the MPI traffic at large processor counts. POP spends 40.8% of the run-time performing small (8 byte) reduction operations. The collective communications used by Open MPI

Table 1. Application Communications Profile with Open MPI's CM Point-To-Point communications

App	# Procs	Ave MPI Time	Message Profile			Top MPI Routines		
		% min,max	# short	# long	#dropped			
			% total	% total	% long	%Tot time	%Tot time	%Tot time
					% min,max			
VH-1	256	16.9% 15.5, 24.7	240 7.4	3000 92.6	890 29.6 9.1, 51.7	Alltoall 15.9	Allreduce 1.0	
GTC	1024	6.1% 2.9, 13.9	6524 43.7	8404 56.3	2130 25.3 7.6, 56.0	Allreduce 4.6	Sendrecv 1.3	Bcast 0.1
POP	256	65.7% 60.6, 70.5	5472986 99.9	5648 0.1	789 13.3 1.8, 93.5	Allreduce 40.8	Waitall 14.4	Isend 5.5
S3D	1024	7.9% 5.5, 9.1	946 0.4	225015 99.6	104020 46.2 25.1, 96.0	Wait 7.2	Allreduce 0.3	Barrier 0.2

use PML level communications for data exchange, and as such the performance of the Point-To-Point communications is one of the factors contributing to overall collective performance.

In addition, Table 1 lists the breakdown of Point-To-Point traffic for the applications. We categorize the data based the communication protocol used; either the short-message protocol used at or below 32K byte cutoff length or the long-message protocol. On average, S3D's, VH-1's, and GTC's Point-To-Point communications are dominated by long messages, with 99.6% of S3D's messages, 92.6% of VH-1's messages, and 56.3% of GTC's messages being long-messages. 46.2%, 29.6%, and 25.3% of the long-messages sent by these respective applications are dropped, and retransmitted once a match is made. While additional time is consumed retreiving the long-message data after the match is made, there does not appear to be a strong correlation between the fraction of long-messages being retransmitted and the overall application performance relative to the CM PML. POP communications are dominated by short-messages, and the long-message protocol is largely irrelevant to its performance in the current set of runs.

5 Conclusions

This paper compares the performance of the Point-To-Point performance of Open MPI's new CM PML with the OB1 PML and with Cray MPI utilizing the Portals communications library. Both micro-benchmarks and full application benchmarks are used. The CM PML is designed to make optimal use of Portals capabilities for providing good application performance at large scale. It provides message injection rates that are comparable to those of Cray MPI and consistently better than those obtained with the OB1 PML. It is superior to both Cray MPI and OB1 with respect to CPU availability, allowing nearly all of the CPU to be available during large message transfers on both the sender and the receiver. CM also has good latency and bandwidth performance curves, comparable with Cray MPI, but superior to the OB1 implementation. With regard to application performance, CM gives slightly better overall performance when compared with Cray MPI, and consistently better performance with respect to OB1.

References

[1] Brightwell, R., Maccabe, A.B.R.R.: Design, implementation, and performance of mpi on portals 3.0. International Journal of High Performance Computing Applications 17(1) (2003)

[2] Barrett, B.W., Brightwell, R., Squyres, J.M., Lumsdaine, A.: Implementation of open mpi on the cray xt3. In: 46th CUG Conference, CUG Summit 2006 (2006)

[3] Graham, R.L., Barrett, B.W., Shipman, G.M., Woodall, T.S., Bosilca, G.: Open mpi: A high performance, flexible implementation of mpi point-to-point communications. Parallel Processing Letters 17(1), 79–88 (2007)

[4] Argonne National Lab.: MPICH2. (http://www-unix.mcs.anl.gov/mpi/mpich2/)

[5] Blondin, J.M., Lufkin, E.A.: The piecewise-parabolic method in curvilinear coordinates. The Astorphysical Journal 88, 589–594 (1993)

[6] Lin, Z., Hahm, T.S., Lee, W.W., Tang, W.M., White, R.B.: Turbulent transport reduction by zonal flows: Massively parallel simulations. Science 281, 1835 (1998)

[7] Dukowicz, J.K., Smith, R., Malone, R.: A reformulation and implementation of the bryan-cox-semter ocean model on the connection machine. J. Atmospheric and Oceanic Tech. 10, 195–208 (1993)

[8] Hawkes, E., Sankaran, R., Sutherland, J., Chen, J.: Direct numerical simulation of turbulent combustion: Fundamental insights towards predictive models. Journal of Physics: Conference Series 16, 65–79 (2005)

[9] Alverson, R.: Red storm. Invited Talk, Hot Chips 15 (2003)

[10] Riesen, R., Brightwell, R., Pedretti, K., Maccabe, A.B., Hudson, T.: The Portals 3.3 Message Passing Interface - Revision 2.1. Technical Report SAND20006-0420, Sandia National Laboratory (2006)

[11] Snell, Q., Mikler, A., Gustafson, J.: In: IASTED International Conference on Intelligent Information Management and Systems (1996)

[12] Doerfler, D., Brightwell, R.: Measuring MPI send and receive overhead and application availability in high performance network interfaces. In: 13th European PVM/MPI Users' Group Meeting, Bonn, Germany (2006)

[13] mpiP: Lightweight, Scalable MPI Profiling http://mpip.sourceforge.net