Parallel Processing Letters

Open MPI: A High Performance, Flexible Implementation of MPI Point-to-Point Communications

Richard L. Graham
National Center for Computational Sciences, Oak Ridge National Laboratory
rlgraham@ornl.gov

and

Brian W. Barrett, Galen M. Shipman, and Timothy S. Woodall
Advanced Computing Laboratory, Los Alamos National Laboratory
{bbarrett,gshipman,woodall}@lanl.gov

and

George Bosilca
Innovative Computing Laboratory, University of Tennessee
bosilca@cs.utk.edu

ABSTRACT

Open MPI's point-to-point communications abstractions, described in this paper, handle several different communications scenarios, with a portable, high-performance design and implementation. These abstractions support two types of low-level communication protocols – general purpose point-to-point communications, like the OpenIB interface, and MPI-like interfaces, such as Myricom's MX library. Support for the first type of protocols makes use of all communications resources available to a given application run, with optional support for communications error recovery. The latter provides a interface layer, relying on the communications library to guarantee correct MPI message ordering and matching. This paper describes the three point-to-point communications protocols currently supported in the Open MPI implementation, supported with performance data. This includes comparisons with other MPI implementations using the OpenIB, MX, and GM communications libraries.

1. Introduction

The high-performance computing arena is currently experiencing several trends that significantly impact the design of message-passing systems. These include a drive towards large process count peta-scale class computing systems, a surge in the number of smaller multi-core clusters being deployed, and the emergence of high performance heterogeneous systems. Developing a message-passing system capable of providing high performance point-to-point communications for this broad range of applications is a challenging task. It requires an ability to efficiently scale with the number of processors, deal with fault scenarios that impact applications running on systems with large component counts, and minimize the impact of supporting heterogeneous environments.

The Open MPI project [3] has developed a flexible abstraction, providing the basis for dealing effectively with a broad set of point-to-point communications scenarios. There have been three methodologies developed code-named OB1 [12] (i.e. OB1 is not short-hand notation for a longer name), DR (Data Reliability), and CM (i.e. CM is not short-hand notation for a longer name). Each of these is an instance of the Point-To-Point Messaging Layer (PML) abstraction. This paper describes these instantiations, provides some performance data obtained with these methods, comparing the performance obtained with different PMLs over the same interconnect, and with other MPI implementations.

A number of implementations of the MPI-1 [8] and MPI-2 [4] standards currently exist. These include LAM/MPI [1], FT-MPI [2], PACX-MPI [7], LA-MPI [5], MPICH [6], MPICH2 [9], the Quadrics version of MPICH [11], Sun's MPI [14], and the Virtual Machine Interface MPI (VMI-MPI) 2.0 from NCSA [10]. Each of these implementations provides its own methodology for point-to-point communications, resulting in a wide range of performance characteristics and capabilities:

- LA-MPI is designed to stripe a single message across identical NICs, different messages across different NIC types, and allows for optional recovery from lost data.
- LAM/MPI's "Component Architecture" enables the user to select from several different point-to-point methodologies at run-time.
- Sun's and Quadrics' MPIs can stripe a single message across multiple available similar Network Interface Cards (NICs).
- MPICH2 has a design intended to scale to very large numbers of processors.
- VMI-MPI is designed to stripe a single message across heterogeneous NICs, giving it the ability to transparently survive the loss of network resources with out any application modification.
- FT-MPI is designed to allow MPI to recover from processor loss with minimal overhead.

The abstractions presented in this paper represent a step forward in the ability of a single MPI implementation to effectively and efficiently handle a broad set of point-to-point communications run-time scenarios, while focusing on high performance. They represent an evolution of ideas first explored in the LA-MPI implementation, then the TEG abstraction [16,15] of Open MPI, and incorporates ideas from LAM/MPI and FT-MPI. The resulting implementation provides a broad and enhanced set of features that include: (a) fault tolerant message passing in the event of transient network faults — in the form of either dropped or corrupt packets — and NIC failures; (b) concurrent support for multiple network types (e.g., Myrinet, InfiniBand, GigE); (c) single message fragmentation and delivery utilizing multiple NICs, including different NIC types, such as Myrinet and InfiniBand; (d) heterogeneous platform support within a single job, including different OS types, different addressing modes (32 vs 64 bit mode), and different endianess; and (e) support for network stacks that provide MPI like semantics with message matching.

The remainder of this paper describes the PML abstraction, and the OB1 DR, and CM implementations of this abstraction. Benchmark results are also presented.

2. Point-To-Point Design

A layered designed is used to implement Open MPI's support for point-to-point communications. The upper layer, accessed directly by the MPI layer is the Point-to-point Messaging Layer (PML). The PML, a Modular Component Architecture (MCA) [13] component, implements all logic for point-to-point MPI semantics such as standard, buffered, ready, and synchronous communication modes, synchronous and asynchronous communications, and the like. MPI message transfers are scheduled by the PML. There are currently three PMLs in active use in the Open MPI code base - OB1, DR, and CM. The TEG PML [16] has been superseded by OB1. These PMLs can be grouped into two categories based on the component architecture used to implement these PMLs, with OB1 and DR forming one group, and CM in a group by itself. The following subsections describes these PMLs, as well as the lower level abstractions developed to support these.

2.1. OB1 *and Data Reliability (*DR*) Point-To-Point Management Layer Components (*PML*s)*

Figure 1 provides a graphical image of Open MPI's Layered architecture. As this figure shows, the OB1 and DR PML design is based multiple MCA frameworks. These PMLs differ in the design features of the PML component itself, and share the lower level Byte Transfer Layer (BTL), BTL 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, as it is used primarily during initialization and BTL selection. These components are briefly described below.

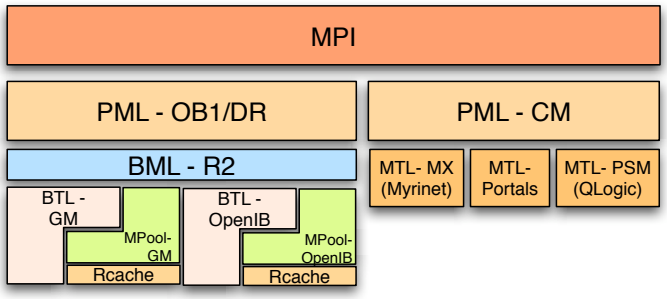

Figure 1: Open MPI's Layered Architecture

The policies these two PMLs implement incorporate BTL specific attributes, such as message fragmentation parameters and nominal network latency and bandwidth parameters, for scheduling MPI messages. These PMLs are designed to provide concurrent support for efficient use of all the networking resources available to a given application run. Short and long message protocols are implemented within the PML, as well as mes-

sage fragmentation and re-assembly. All control messages (ACK/NACK/MATCH) are also managed by the PML. The benefit of this structure is a separation of the high-level (e.g., MPI) transport protocol from the underlying transport of data across a given interconnect. This significantly reduces both code complexity and code redundancy while enhancing maintainability, and provides a means of building other communications protocols on top of these components.

While the OB1 and DR PMLs share a many common features, they differ in the sort of communications they support. The OB1 PML is designed for low-latency, high bandwidth communications, and assumes that all communications will succeed, or directly inform the calling layer of failure. The only way communication failure is detected is via error code propagation from low-level communications libraries. This component possesses a limited to a simple set of corrective actions for these failures, such as attempts to re-establish network connectivity, or else it just terminates communications. The OB1 PML is designed to deliver scalable, high-performance communications. It does not add overhead needed to support general fault recovery, such as watch-dog timer maintenance.

The DR PML is aimed to provide high performance scalable point-to-point communications in the context of potential network failures. This includes transient network failures, such as data corruption, dropped packets, and temporary network disconnects. More catastrophic network failures, such as partial loss of network connectivity between two given end-points, may also be handled, if alternative network paths exist between these two end-points. These network failures are dealt with at the PML level, without the need for the application to change. A later paper will describe this PML in detail.

The common MCA frameworks used in support of the OB1 and DR PMLs are described briefly below.

MPool The memory pool provides memory allocation/deallocation and registration / de-registration services. For example, InfiniBand requires memory to be registered (physical pages present and pinned) before send/receive or RDMA operations can use the memory as a source or target. Separating this functionality from other components allows the MPool to be shared among various layers. For example, MPI_ALLOC_MEM uses these MPools to register memory with available interconnects.

Rcache The registration cache allows memory pools to cache registered memory for later operations. When initialized, MPI message buffers are registered with the Mpool and cached via the Rcache. For example, during an MPI_SEND the source buffer is registered with the memory pool and this registration may be then be cached, depending on the protocol in use. During subsequent MPI_SEND operations the source buffer is checked against the Rcache, and if the registration exists the PML may RDMA the entire buffer in a single operation without incurring the high cost of registration.

BTL The BTL modules expose the underlying semantics of the network interconnect in a consistent form. BTLs expose a set of communication primitives appropriate for both send/receive and RDMA interfaces. The BTL is not aware of any MPI semantics; it simply moves a sequence of bytes (potentially non-contiguous)

across the underlying transport. This simplicity enables early adoption of novel network devices and encourages vendor support. There are several BTL modules currently available; including TCP, Myrinet/GM, Myrinet/MX, Cray Portals, Shared Memory (SM), Mellanox VAPI, and OpenIB VAPI.

BML The BML acts as a thin multiplexing layer, allowing the BTLs to be shared among multiple upper layers. Discovery of peer resources is coordinated by the BML and cached for multiple consumers of the BTLs. After resource discovery, the BML layer may be safely bypassed by upper layers for performance. The current BML component is named R2.

This class of PMLs makes use of some well-designed primitive MCA components to build very portable and extensible communications protocols. In particular, implementation of a single set of components for a given communications stack is all that is needed to implement this class of PML communications protocols - OB1 and DR, in the current implementation.

2.2. CM *Point-To-Point Management Layer Component (*PML*)*

The CM PML is designed to provide an MPI interface directly utilizing APIs that expose matching send/receive semantics capable of supporing MPI communication protocols. As the matching logic is implemented in the underlying network library, the CM component is much smaller than the OB1 or DR components. CM handles memory management for requests and buffer management for MPI's buffered sends. The other aspects of MPI's point-to-point semantics are implemented by the Matching Transport Layer (MTL) framework, which provides an interface between the CM PML and underlying network library. Currently there are three implementations of the MTL, for Myricom's MX library, QLogic's InfiniPath library, and the Cray Portals communication stack.

The CM PML pushes the MPI matching logic as close to the network hardware as possible, providing the opprotunity for Open MPI to provide latency and bandwidth almost identical to the native network interface. Unlike the OB1 and DR components, CM allows message matching and transfer to occur asynchronously, without the application entering the MPI layer and without progress threads in the MPI layer.

The CM component, however, is unable to provide MPI-level message striping or use different interconnects for different endpoints. It is assumed that interfaces that provide high-level matching logic implement shared memory optimizations within the lower-level library. Further, due to an issue with the Open MPI datatype engine and any-source receives, the CM PML does not support heterogeneous environments. In many situations, however, the possibility of better performance and the asynchronous progress of CM outweigh its disavantages.

3. Results

A simple ping-pong benchmark has been used to gather bandwidth and latency performance data. Open MPI performance data is collected for the OB1, DR, and CM PMLs, and compared, as appropriate, with the same performance data collected with Myricom's MPICH MX and MPICH-GM, MVAPICH MPI, and QLogic's InfiniPath MPI.

3.1. Experimental Setup

Results for Myrinet/MX, Myrinet/GM, and OpenIB were generated on a small Opteron cluster, with two 2.0 GHz dual core Opteron processors and 4 GB of memory per node. Each machine contains two Mellanox InfiniHost III EX InfiniBand HCAs on dedicated 16x PCI Express busses and a Myrinet PCI-X D NIC in a 133 MHz PCI-X slot. The Mellanox InfiniBand HCAs are connected via a SilverStorm 9120 DDR switch and the Myrinet NICs are connected via a Myrinet 2000 switch, with all nodes used in testing connected to the same line card. Results for the InfiniPath InfiniBand adapter were obtained using Qlogic's CBC test cluster. Each machine has the Iwill DK8-HTX motherboard with two 2.6 GHz Opteron processors. The InfiniPath adapter is connected to the motherboard via the Hyper Transport 1.0 interface.

3.2. Latency

The latency results are reported in Table 1, and are the half round trip time to transfer one byte of MPI data between two processes. One byte was used, rather than zero bytes, as at zero byte one of the implementations we compared against exhibited anomalously high latencies.

MPI Implementation	PML	MTL	BTL	Average 1 Byte Latency
Open MPI	CM	MX		3.26
MPICH MX				2.97
Open MPI	OB1		OpenIB	2.99
Open MPI	DR		OpenIB	6.21
MVAPICH MPI				2.84
Open MPI	OB1		GM	7.59
Open MPI	DR		GM	12.10
MPICH GM				7.54
Open MPI	CM	PSM		1.45
InfiniPath MPI				1.29

Table 1: Two node Ping-Pong latency in μ-sec (one-way).

The OB1 latencies were measured for the OpenIB and GM BTLs. The OpenIB latency was measured at 2.99 μ-sec, slightly above the MVAPICH MPI latency which was measured at 2.84 μ-sec, and the GM latency was measured at 7.59 μ-sec, very slightly above the 7.54 μ-sec measured with Myricom's MPICH-GM. Open MPI's OB1 latencies are comparable to those of both MVAPICH and MPICH-GM MPI implementations which are tailored to their respective communication protocols. This data indicates that Open MPI's general purpose design meets the objective of providing low-latency communications.

The DR latencies were measured for the OpenIB and GM BTLs. The OpenIB latency was measured at 6.21 μ-sec, an increase of about 3 μ-sec, or about 107%, above the OB1 latency. The GM latency was measured at 12.10 μ-sec, about 4.5 μ-sec, or 60%, above that measured with the GM OB1 PML is expected, as the reliability protocol

adds acknowledgments to ensure correct data delivery, as well as optional end-to-end data integrity checks to detect data corruption. We are unaware of any other MPI implementation that provides such capability using os-bypass capabilities.

Finally, the CM latencies were measured for QLogic's InfiniPath MTL, and Myricom's MX MTL. The InfiniPath MTL gave a latency of 1.45 μ-sec compared with a latency of 1.29 μ-sec with InfiniPath MPI. The MX MTL came in with a latency of 3.26 μ-sec, which is slightly higher than that produced by MPICH MX. Again, Open MPI's CM latencies are comparable to those of these two MPI implementations which are tailored to their respective communication protocols.

3.3. Bandwidth

The bandwidths measured with the OB1 and the CM components are displayed in Figures 2 and 3, respectively. As these figures indicate, the Open MPI implementation nearly saturates available network bandwidth at large message sizes, and over the entire range of message sizes measured, performs as well as, and in some instances better than, the other MPI implementations. However, the OB1 implementation, will aggregate data across multiple network connections, when present, thus greatly increasing the measured ping-pong bandwidth.

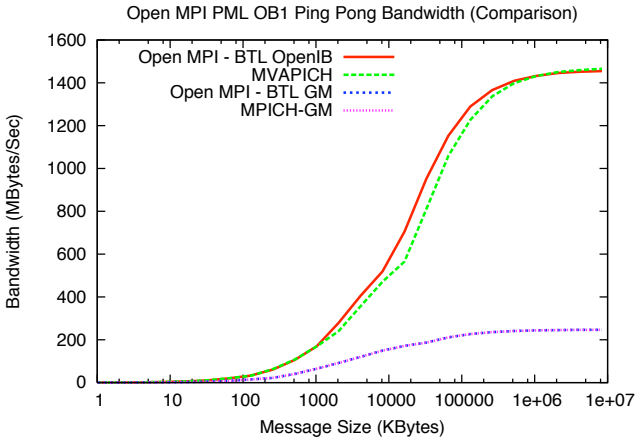

Figure 2: OB1 Ping-Pong Bandwidth (MB/Sec)

Bandwidths measured with the DR OpenIB and GM BTLs are displayed in Figure 4. At small message sizes, where the network transmission time dominates the overall communications time, the bandwidths measured with the DR components are similar to those measured with the OB1 components. At larger message sizes, where the time to traverse the data and generate a checksum at both the sender and the receiver gets large, these costs start to take a large portion of the overall transmission time, significantly reducing the measured bandwidth. With the OpenIB component, which uses the Network Interface Card's RDMA capabilities, this translates to a reduction of about 30% in measured peak bandwidth. With the GM network stack, DR and OB1 obtain similar bandwidth profiles. This is not surprising, given that the OB1

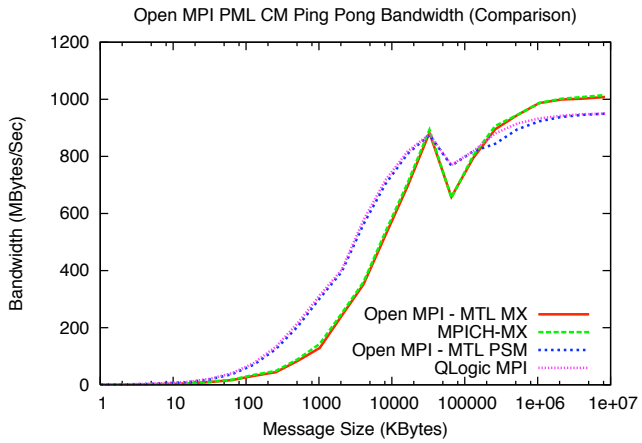

Figure 3: CM Ping-Pong Bandwidth (MB/Sec)

GM BTL uses a memory copy to transfer data into registered GM buffers, and the checksum used to protect the data is relatively inexpensive, and pipelined with the memory copies. When one uses a similar copy-in/copy-out protocol with the OpenIB BTL, similar performance characteristics are also observed.

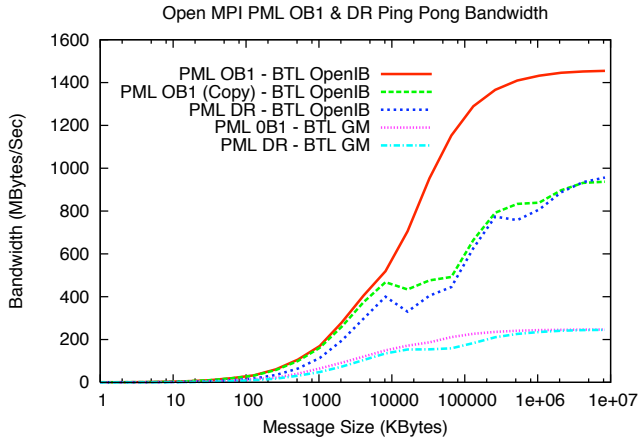

Figure 4: DR Ping-Pong Bandwidth (MB/Sec)

4. Summary

This paper discusses the point-to-point design used in Open MPI to handle, within a single library build, a wide range of network stacks. In addition the ability to recover transparently from several types of network errors is implemented, thus providing a means to increase applications mean time to failure. As the latency and bandwidth

measurements show, the performance obtained with this library is very good. Future point-to-point work will involve further latency and bandwidth optimizations, with most of the effort aimed at honing the DR component and overall thread safety.

Acknowledgments

This work was supported by National Science foundation grant EIA-9972889. Project support was provided through ASCI/PSE, the Los Alamos Computer Science Institute, and the Center for Information Technology Research (CITR) of the University of Tennessee. Los Alamos National Laboratory is operated by Los Alamos Security, LLC for the United States Department of Energy's National Nuclear Security Administration. LA-UR-06-7990.

References

[1] G. Burns, R. Daoud, and J. Vaigl. LAM: An Open Cluster Environment for MPI. In *Proceedings of Supercomputing Symposium*, pages 379–386, 1994.

[2] Graham E. Fagg, Edgar Gabriel, Zizhong Chen, Thara Angskun, George Bosilca, Antonin Bukovski, and Jack J. Dongarra. Fault tolerant communication library and applications for high perofrmance. In *Los Alamos Computer Science Institute Symposium*, Santa Fee, NM, October 27-29 2003.

[3] E. Garbriel, G.E. Fagg, G. Bosilica, T. Angskun, J. J. Dongarra J.M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, R.H. Castain, D.J. Daniel, R.L. Graham, and T.S. Woodall. Open MPI: goals, concept, and design of a next generation MPI implementation. In *Proceedings, 11th European PVM/MPI Users' Group Meeting*, 2004.

[4] A. Geist, W. Gropp, S. Huss-Lederman, A. Lumsdaine, E. Lusk, W. Saphir, T. Skjellum, and M. Snir. MPI-2: Extending the Message-Passing Interface. In *Euro-Par '96 Parallel Processing*, pages 128–135. Springer Verlag, 1996.

[5] R. L. Graham, S.-E. Choi, D. J. Daniel, N. N. Desai, R. G. Minnich, C. E. Rasmussen, L. D. Risinger, and M. W. Sukalksi. A network-failure-tolerant message-passing system for terascale clusters. *International Journal of Parallel Programming*, 31(4), August 2003.

[6] W. Gropp, E. Lusk, N. Doss, and A. Skjellum. A high-performance, portable implementation of the MPI message passing interface standard. *Parallel Computing*, 22(6):789–828, September 1996.

[7] Rainer Keller, Edgar Gabriel, Bettina Krammer, Matthias S. Mueller, and Michael M. Resch. Towards efficient execution of MPI applications on the grid: porting and optimization issues. *Journal of Grid Computing*, 1:133–149, 2003.

[8] Message Passing Interface Forum. MPI: A Message Passing Interface. In *Proc. of Supercomputing '93*, pages 878–883. IEEE Computer Society Press, November 1993.

[9] Mpich2, argonne, http://www-unix.mcs.anl.gov/mpi/mpich2/.

[10] S. Pakin and A. Pant. VMI 2.0: A dynamically reconfigurable messaging layer for availability, usability, and management. In *Proceedings of The 8th International Symposium on High Performance Computer Architecture (HPCA-8)*, Cambridge, MA, February 2002.

[11] Quadrics, llc web page, http://www.quadrics.com/.

[12] G.M. Shipman, T.S. Woodall, R.L. Graham, A.B. Maccabe, and P.G. Bridges. Infiniband scalability in open mpi. In *Proceedings, 20th IEEE International Parallel & Distributed Processing Symposium*, 2006.

[13] Jeffrey M. Squyres and Andrew Lumsdaine. The component architecture of open MPI: Enabling third-party collective algorithms. In Vladimir Getov and Thilo Kielmann, editors, *Proceedings, 18th ACM International Conference on Supercomputing, Workshop on Component Models and Systems for Grid Applications*, pages 167–185, St. Malo, France, July 2004. Springer.

[14] Sun, llc web page, http://www.sun.com/.

[15] T.S. Woodall, R.L. Graham, R.H. Castain, D.J. Daniel, M.W. Sukalsi, G.E. Fagg, E. Garbriel, G. Bosilica, T. Angskun, J. J. Dongarra, J.M. Squyres, V. Sahay, P. Kambadur, B. Barrett, and A. Lumsdaine. Open MPI's TEG point-to-point communications methodology : Comparison to existing implementations. In *Proceedings, 11th European PVM/MPI Users' Group Meeting*, 2004.

[16] T.S. Woodall, R.L. Graham, R.H. Castain, D.J. Daniel, M.W. Sukalsi, G.E. Fagg, E. Garbriel, G. Bosilica, T. Angskun, J. J. Dongarra, J.M. Squyres, V. Sahay, P. Kambadur, B. Barrett, and A. Lumsdaine. TEG: A high-performance, scalable, multi-network point-to-point communications methodolog. In *Proceedings, 11th European PVM/MPI Users' Group Meeting*, 2004.