Implementation of Open MPI on Red Storm

Brian W. Barrett, Jeffrey M. Squyres, and Andrew Lumsdaine

September 30, 2005

Abstract

The Open MPI project provides a high quality MPI implementation available on a wide variety of platforms. This technical report describes the porting of Open MPI to the Cray Red Storm platform (as well as the Cray XT3 platform). While alpha quality, the port already provides acceptable performance. Remaining porting work and future enhancements are also discussed.

1 Introduction

The Open MPI [10] project seeks to provide a high quality MPI implementation for a wide variety of hardware. The performance on commodity Linux clusters with high speed interconnects is well established [21]. In addition to Linux, Open MPI also supports a variety of commodity and commercial operating systems, including AIX, Mac OS X, and Solaris. Until recently, Open MPI had not been ported to tightly integrated supercomputing environments.

Cray Red Storm is a distributed memory massively parallel architecture designed to scale to tens of thousands of processors. The platform was originally designed for Sandia National Laboratories to fulfill their large-scale high performance computing needs. Cray has since productized the machine as the Cray XT3, a number of which have been delivered to research computing centers across the United States.

This technical report details the porting of Open MPI to Red Storm, providing details on both the runtime environment changes necessary to support such a unique platform and the development of a communication channel to support Red Storm's high-speed interconnect. Performance results from both micro-benchmarks and real-world benchmarks are provided. As the port is currently in an alpha stage, the future work section provides a detailed discussion of development required to make Open MPI a first class MPI implementation on Red Storm.

2 Background

This section provides a brief overview of both Open MPI [3, 7, 10, 19, 21, 22] and the Cray Red Storm platform [1, 4, 6]. A familiarity with both is assumed for the remainder of this report.

2.1 Open MPI

Open MPI is the result of a collaboration between the authors of the LAM/MPI [5, 20], LA-MPI [2, 12], and FT-MPI [8, 9] implementations of the MPI standard. Open MPI supports the MPI-1 standard [15, 18] and much of the MPI-2 standard [11, 13]. One-sided support is currently missing, but should be available during FY'06. Open MPI is designed to offer high performance and scalability on a variety of platforms. In order to support a wide range of interconnections and platforms, Open MPI utilizes a low overhead component infrastructure, the MCA [3, 19].

Open MPI leverages the Open Run-Time Environment (ORTE) [7] for run-time support. ORTE was developed as part of the Open MPI project, but it is slowly becoming an independent project. ORTE utilizes a number of component frameworks for providing process control and out-of-band messaging between processes. ORTE currently provides out-of-band messaging over TCP, and process control us-

ing RSH/SSH, PBS TM, SLURM, Apple XGrid, and IBM POE.

2.2 Cray Red Storm

The Red Storm machine at Sandia National Laboratories in Albuquerque, New Mexico currently consists of 10,368 processors. Each node contains a single 2.0 GHz Opteron CPU with 2 GB of main memory and a Cray SeaStar NIC/router attached via HyperTransport. The network is a 27x16x24 mesh topology, with 2.0 GB/s bidirectional link bandwidth and 1.5 GB/s bidirectional node bandwidth. The nearest neighbor NIC to NIC latency is specified to be 2 μsec, with 5 μsec worst case latency.[1] The compute nodes run the Catamount lightweight kernel, a follow-on to the Cougar/Puma [17] design used on ASCI Red [14]. The I/O and administrative nodes run a modified version of SuSE Linux.

The Cray-designed SeaStar [1] communication processor / router is designed to offload network communication from the main processor. It provides both send and receive DMA engines, a 500MHz PowerPC 440 processor, and 384 KB of scratch memory. Combined with the Catamount lightweight kernel, the SeaStar is capable of providing true OS-bypass communication.

The Red Storm platform utilizes the Portals 3.3 communication interface [4], developed by Sandia National Laboratory and the University of New Mexico for enabling scalable communication in a high performance computing environment. The Portals interface provides true one-sided communication semantics. Unlike traditional one-sided interfaces, the remote memory address for an operation is determined by the target, not the origin. This allows Portals to act as a building block for high performance implementations of both one-sided semantics (Cray SHMEM) and two-sided semantics (MPI-1 send/receive). Matching and communication processing on Red Storm is currently done in interrupt handlers running on the main Opteron CPU. In the near future, it is expected that matching and most communication processing will be offloaded into the firmware running on the SeaStar's PowerPC 440 CPU, greatly lowering latency.

The Cray XT3 commercial offering is nearly identical to the Red Storm machine installed at Sandia. The notable difference is that while the Red Storm communication topology is a 3-D mesh, the XT3 utilizes a 3-D torus configuration. The difference is to allow a significant portion of the Red Storm machine to switch between classified and unclassified operation. For the remainder of the paper, the term Red Storm refers to both the Sandia Red Storm machine and the Cray XT3, unless otherwise noted.

3 Run-Time Environment

The initial concern for porting Open MPI to the Red Storm platform was the Open Run-Time Environment and it's ability to integrate with the YOD process control system utilized on Red Storm. The `yod` command will be used to start the MPI job instead of the traditional `mpirun` command, removing the centralized startup mechanism available on other platforms supported by Open MPI. `yod` handles process startup, process monitoring, and standard I/O in a scalable fashion. Node allocation and communication setup is handled by the Cray run-time environment.

Many components in ORTE had configure scripts added or modified to check for functionality that is not available on the Red Storm platform. For example, the TCP OOB component now checks for TCP support and the RSH PLS checks for the `fork()` system call before building. These checks are not strictly Red Storm specific, but were motivated by the need to support the platform.

Complicating the porting was that the Red Storm environment would be the first environment Open MPI would encounter that did not provide a TCP stack available for out-of-band communication — the compute nodes only provide the Portals interface for communication. Portals does not require any out-of-band communication for process wire-up and the run-time system is not designed to support the MPI-2 dynamic process chapter, so it was decided that the initial port should forego out-of band communication.

[1] The Red Storm machine does not currently provide these performance characteristics — nearest neighbor latency is closer to 5 μsec. Firmware performance tuning is ongoing and a new revision of the SeaStar should greatly improve latency.

This had a trickle down effect on other aspects of the run-time environment, discussed below.

As there are no available OOB components on Red Storm, the OOB Run-Time Messaging Layer (RML) is not available. A Red Storm specific RML communication channel — CNOS — was added to support the barrier functionality available as part of the compute node runtime support (both as `rml_bcast()` with a `NULL` payload and `rml_barrier()`). No other communication functionality is support by the CNOS RML. Blocking communication calls return an error immediately (`ORTE_ERR_NOT_IMPLEMENTED`), while non-blocking calls succeed, but the message callback is never fired.

A new General Purpose Registry (GPR) component — NULL — was added in order to handle the lack of a central "daemon" process that is found in most Open MPI environments. The NULL GPR component will return an error on all immediate functions (`gpr_put()`, for example) and will succeed for all non-blocking calls, but the callback will never be fired. The MPI layer registers a number of non-blocking GPR triggers, but does not use blocking GPR calls. By setting reasonable defaults before registering the non-blocking call, it is possible to execute with the NULL GPR component — the callback never fires, so the default values are used. For example, the universe size MPI parameter is initially set to be the size of MPI_COMM_WORLD. A GPR trigger is registered to update the value if there is a larger universe size, but if the trigger is never fired, a reasonable value still exists.

One new framework was required in order to support Red Storm. The System Discovery Service (SDS) was implemented to provide a component infrastructure for providing the starting process both its current ORTE name[2] and the list of names for the current jobid. As there is a one to one mapping between jobs and MPI_COMM_WORLDs, it is possible to determine both the rank in and the size of MPI_COMM_WORLD. Generally, this information is provided by the starting agent, almost always through environment variables set by the process starter (`mpirun`, MPI_COMM_SPAWN, etc.). Previously, this information was provided by code in the base library of the NS framework. Moving to a new component framework allowed for conditional compilation of the components which provide process naming information. The CNOS SDS component, added to support Red Storm, utilizes the information provided by the Cray run-time environment in order to provide process group information.

[2]All ORTE processes are assigned a unique name. Currently, the name consists of a triple of (cellid, jobid, vpid), although this may change in the near future.

4 Communication

Open MPI provides a layered architecture for point-to-point communication, shown in Figure 1. The lowest layer, the Byte Transport Layer (BTL) provides active-message-like send and true RDMA put/get semantics. The BTL interface is extremely small, consisting of eleven functions and has no concept of MPI structures or semantics. The BTL Management Layer (BML) provides scheduling and multiplexing of BTL instances, allowing a single BTL to be shared between multiple higher level protocols. The PML provides MPI point-to-point semantics, and may use the BML/BTL design if desired. The OB1 PML provides message striping and fragmenting over the BTL components. Ongoing research seeks to add fault tolerance and message reliability to OB1.

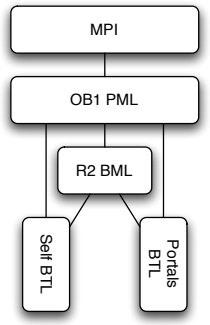


Figure 1: Component design for point-to-point communication.

Given the ease of mapping Portals functionality to MPI semantics, it was not clear whether it was better

to implement Portals communication support at the PML or BTL level. The decision was made to implement Portals support at the BTL level for two reasons: it was believed that performance would be competitive with a PML implementation and the MPI-2 one-sided support utilizes the BTL interface, bypassing the PML interface. If the Red Storm implementation is to support the MPI-2 one-sided chapter, it will require either a Portals BTL or a Portals-specific one-sided implementation. Therefore, it was decided to implement Portals support at the BTL level to reduce the amount of Red Storm-specific code.

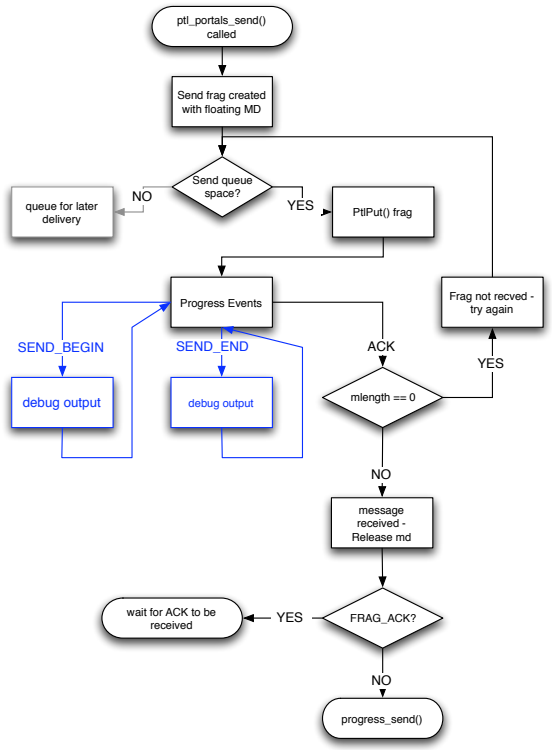


Figure 2: Flow chart of the `btl_send()` logic for Portals.

The Portals BTL provides implements both send and RDMA communication. The send communication is implemented utilizing a design similar the method used for unexpected short messages described in [16]. Messages are put into pre-posted receive buffers. The BTL tag is sent to the receiving process as the `user_data` option of the `PtlPut()` function, allowing messages to be sent with no BTL specific headers. A complete flowchart of the send logic utilized by the Portals BTL is provided in Figure 2. The number of outstanding send fragments is limited to ensure there is space in the send message event queue to receive any pending acknowledgments. Because send fragments may be dropped during periods of heavy traffic and need to be retransmitted, the calling component is not notified of completion until a successful acknowledgment is received. The OB1 PML can return from MPI send functions before the completion notification is received and still maintain MPI semantics, so this is not a performance concern in most situations. Portals vectored puts and the `BTL_SEND_IN_PLACE` option are used so that user data is only copied into a temporary buffer if it is non-contiguous. Otherwise, the only memory copies necessary within the PML or BTL are for the PML header protocols.

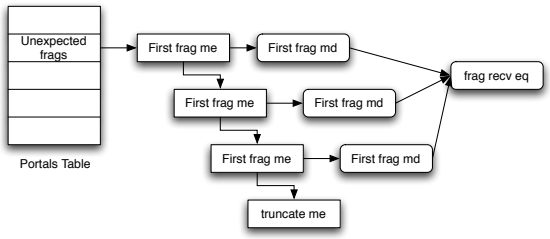


Figure 3: Match list and memory descriptor configuration for receiving message fragments.

Receiving send messages is done via a set of large memory segments (approximately 1 MB each) attached to the Open MPI receive portal table entry. Figure 3 provides an outline of the Portals structures for receiving `btl_send()` fragments. A "reject" match entry / memory descriptor sits as the last entry in the match list, which has no associated event queue and truncates all messages to 0 byte length. This allows the sending process to be notified (via the standard Portals acknowledgment mechanism) that a message was not received and should be retransmitted. The receive message descriptors share an event

queue with RDMA operations. The initial design utilized three event queues (one for send, one for receives, and one for RDMA operations), but it was reduced to two event queues (one for send and one for receives/RDMA) because there is a small overhead for each event queue polled, increasing latency. When events are received from the shared receive/RDMA queue, the `user_data` field of the event structure is used to determine whether the event was for a receive or RDMA operation.

RDMA operations are largely a one-to-one mapping between the BTL put / get functions and the `PtlPut()` / `PtlGet()` functions. Memory descriptors are created during `btl_prepare_src()` and `btl_prepare_dest()`, and match entries are created and attached to a portals table entry for RDMA operations. The semantics of the BTL descriptor sharing means that the target of the RDMA operation can pass a unique 64 bit match key to the origin, allowing the use of Portals matching for RDMA messages. Because there is currently not enough information passed to `btl_prepare_src()` to determine whether the descriptor will be the origin of a put or the target of a get operation, the match entry is always created (the `btl_prepare_dest()` function has the symmetric problem). This causes a slight overhead in creating descriptors for RDMA operations, but that latency is generally masked by the eager send and the RDMA protocol setup used by the OB1 PML.

5 Results

5.1 Alpha Requirements

The goal for FY'05 was to provide an alpha quality release of Open MPI for the Red Storm platform. Alpha quality was defined as successfully executing the non-performance C tests in the Intel Test suite that is part of the Open MPI test repository and provide reasonable performance comparable to MPICH.[3] Subversion revision number 7507 from the Open MPI trunk was used to successfully run the Intel test suite on a 28 node test cage at Sandia National Laboratory, Albuquerque.[4]

5.2 Performance

Open MPI's support of Red Storm currently lacks some features required to achieve performance comparable to either Sandia's MPICH 1.2.6 or Cray's MPICH-2 implementations, namely hardware message matching and zero-copy receives. Implementation improvements are discussed in Section 6.1.

5.2.1 Latency

Figure 4 shows the best case message latency for the three MPI implementations tested and direct Portals communication. The lack of hardware message matching has a clear effect on latency. There is also a small amount of overhead for Open MPI due to the header required for software matching (there is no extra message header for MPICH 1.2.6 or MPICH-2). It is possible to save .25 - .75 μsec on the zero byte latency by implementing a copy protocol that reuses small message memory descriptors, but the decision was made to delay such implementation until after hardware matching was implemented (see Section 6 for more information).

Implementation	0 Byte	1 Byte
Portals	5.15 μsec	5.20 μsec
MPICH	5.49 μsec	5.78 μsec
MPICH-2	7.19 μsec	7.90 μsec
Open MPI	9.63 μsec	12.50 μsec

(a)

Figure 4: Latency for zero and one byte messages using mpi-ping.

The drastic difference between 0 and 1 byte latency for Open MPI is due to a combination of factors. There is a small amount of overhead because 0 byte messages follow an optimized communication path and do not initialize the datatype engine. Approximately 2 μsec of the latency difference is due to

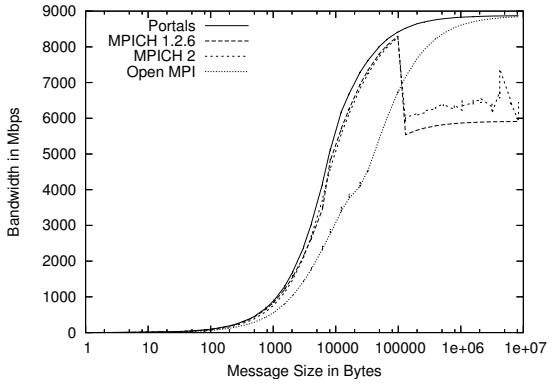
[3] The version of the Intel test suite in the test repository is slightly modified from the stock version, correcting some bugs and tests that do not conform to the MPI standard.

[4] Due to two node failures, only 26 nodes were available for testing.

the Portals implementation running on Red Storm. Portals is able to handle a 16 byte payload in one interrupt, but any more requires at least two interrupts to properly handle. The match header in Open MPI is currently 16 bytes, so a 0 byte message can be sent in one interrupt, but a 1 byte message will take two interrupts to send.

5.2.2 Bandwidth

Figure 5 shows the achievable unidirectional bandwidth achievable on Red Storm for the three available MPI implementations and direct Portals communication. For very small messages, the extra latency due to software matching adversely effects bandwidth. For messages up to 64 KB, the lower bandwidth than MPICH appears to be due to a combination of software message matching and the need to copy messages from the receive buffers into user memory. At 64 KB, an RDMA get protocol is used. The first portion of the message, along with the source BTL descriptor for the RDMA get is sent. Once the message is matched, the receiving side performs a `PtlGet()` to receive the remainder of the message. A completion acknowledgment is then sent from the receiver to the sender. While peak bandwidth is comparable to MPICH, the bandwidth for medium sized messages (64 KB - 238 KB) is lower, due to the latencies added by the extra protocol messages.



(b)

Figure 5: NetPIPE bandwidth on Red Storm.

The noisy bandwidth measurements for large messages when using MPICH 1.2.6 and MPICH-2 on the Red Storm machine is likely due to unexpected receives. Both implementations are designed to provide the best performance for small messages and large expected messages, at the cost of unexpected large message performance. While there are options to reduce the cost of unexpected receives, they generally come at the expense of either expected receives or true asynchronous progress.

6 Future Work

The 2005 Open MPI milestone to support the Red Storm architecture required an "alpha quality" MPI implementation. While we believe this goal has been met, there is still much remaining before the port can be defined as complete. This section discusses the remaining issues and possible implementation options.

6.1 Portals-level matching

Currently, all message matching logic is performed in the OB1 PML, at the MPI software layer. MPICH 1.2.6 and MPICH-2 on Red Storm utilizes the matching capabilities of the Portals interface for message matching, resulting in much lower short message latencies and better medium-sized message bandwidth. We believe that it is necessary to push matching into the Portals layer to achieve better performance. Two proposals exist for pushing message matching into the portals layer: extend the BTL interface to support hardware matching or developing a PML for Portals.

6.1.1 BTL with NIC matching

The current OB1 PML / BTL interface does all message matching inside the OB1 interface. Message fragments transmitted via `btl_send()` are unexpected messages and not received directly into user buffers. In order to allow for hardware-based receives, the BTL interface would have to be extended to notify BTLs to expect a receive. In addition, the PML would have to be modified (or possibly a new PML

added) to only allow one BTL to send match fragments, and to take advantage of the hardware matching. The advantage of this approach is a high reuse of existing code. The semantics of the BTL interface could remain largely unchanged, with the addition of some type of matching key on `btl_send()` operations and a receive function. In addition, the one-sided interface could be extended to make use of the hardware matching features, enhancing performance of the MPI-2 one-sided interface. There are some disadvantages to this approach — the RDMA setup protocol required by the BTL may be too much overhead to deliver performance comparable to MPICH and the polling for unexpected messages may require a third event queue, slightly increasing best case latency. Because the exact requirements of such a design are unknown, no reasonable estimate of implementation time is available.

6.1.2 Portals PML

Another implementation option for improving point-to-point performance is to implement a Portals-specific PML interface. This approach would allow for all optimizations described in the various literature on MPI implementations for Portals. All expected sends and receives should be able to occur with zero copies (provided the datatype used is somewhat contiguous), increasing bandwidth for mid-sized messages. The disadvantage of this approach is a complete rewrite of the communication code — the BTL code that is currently operational would not be reusable in this scenario. Given the easy of implementing most MPI semantics with the Portals API, it is not expected to take more than a man-month to implement a reasonably optimized Portals PML.

6.2 Small message optimization

Currently, all short messages in the Portals BTL are using iovecs, with the calling layer's header in the first iovec entry (space for which is part of the BTL descriptor), and the second entry pointing directly to the user buffer.[5] A memory descriptor is created during `btl_alloc()` / `btl_prepare_src()` and unlinked during `btl_desc_free()`. Memory descriptor creation and deletion requires a kernel trap, so it may be faster to pre-allocate a small number of small memory descriptors for use in sending very small messages, copying the data into the buffers rather than sending in place. If a Portals PML is implemented, a similar optimization is available, including creating memory descriptors for the entire memory space and sending small messages without ever creating memory descriptors.

[5] Unless the user buffer is non-contiguous, in which case the data will be memory copied into a send fragment.

6.3 MPI topology component

The Red Storm machine provides access to the X,Y,Z coordinates on the mesh to each process. From this, it is possible during MPI_INIT to determine the topology of the entire job. The TOPO framework allows for intelligent implementations of the MPI topology functions. Currently, the UNITY component is used on Red Storm, providing no real topology information to the MPI process. It is believed that similar information is available on the Cray XT3.

6.4 Topology aware collectives

While not strictly an optimization specific to the Red Storm / XT3 architectures, the collective routines currently in use on the platform are generic routines that have no awareness of topology. Given the low latencies of nearest neighbor communication on Red Storm, collectives could see great improvements if made aware of network topology. There is also discussion among Sandia researchers of pushing some collectives logic into the Portals API — if such an optimization was made available, a new TOPO component for Portals would be required.

6.5 MPI-2 I/O

Open MPI uses the ROMIO package from Argonne National Laboratory for MPI-2 I/O routines. The stock ROMIO 1.2.4 included in version 1.0 of Open MPI does not properly support cross-compiling, so it is currently disabled on Red Storm. It also does not have direct support for the Red Storm parallel

file system, although it appears that an ADIO implementation is available that can be integrated into Open MPI's release of ROMIO.[6]

6.6 Fortran Bindings

The configure script for Open MPI's Fortran bindings do not support cross compiling, so current builds of Open MPI on Red Storm lack Fortran support. The tests that require cross compiling are to determine size and padding of Fortran types. Unfortunately, while Autoconf has support for determining size and alignment of C types when cross compiling, it does not have such support for Fortran. In fact, it is not clear that it is possible to determine size and alignment of Fortran types without running a test program. The current best solution for this problem is to provide a small test program for determining the required information, which is given to the configure script. Support for Fortran on the Red Storm platform is planned for FY'06. It is believed that once configuration issues are resolved, there should be no remaining issues with the Fortran bindings on Red Storm.

7 Conclusions

Open MPI has shown the ability to operate in tightly integrated supercomputing environments, such as Red Storm. Performance is adequate, and can be improved greatly by the addition of hardware matching and the reduction of protocol overhead in the OB1 or by the addition of a Portals-specific PML component. A small number of build and implementation issues must be resolved before the port can be called complete, but all are minor in scope.

Thanks

This work was supported by a grant from the Lilly Endowment, National Science Foundation grant ANI-0330620, and University of California (Los Alamos National Lab) subcontract number 15043-001-05. Thank you to Ron Brightwell for providing both advice and machine time for the development of Open MPI on Red Storm.

[6] Pending the inevitable licensing issues.

References

[1] Robert Alverson. Red storm. In *Invited Talk, Hot Chips 15*, 2003.

[2] Rob T. Aulwes, David J. Daniel, Nehal N. Desai, Richard L. Graham, L. Dean Risinger, Mitchel W. Sukalski, Mark A. Taylor, and Timothy S. Woodall. Architecture of LA-MPI, a network-fault-tolerant mpi. In *Los Alamos report LA-UR-03-0939, Proceedings of IPDPS*, 2004.

[3] B. Barrett, J. M. Squyres, A. Lumsdaine, R. L. Graham, and G. Bosilca. Analysis of the component architecture overhead in open mpi. In *Proceedings, 12th European PVM/MPI Users' Group Meeting*, Sorrento, Italy, September 2005.

[4] Ron Brightwell, Tramm Hudson, Arthur B. Maccabe, and Rolf Riesen. The portals 3.0 message passing interface. Technical Report SAND99-2959, Sandia National Laboratories, 1999.

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

[6] W. Camp and J.L. Tomkins. Thor's hammer: The first version of the red storm mpp architecture. In *Proceedings of the SC 2002 Conference on High Performance Networking and Computing*, Baltimore, MD, 2002.

[7] R. H. Castain, T. S. Woodall, D. J. Daniel, J. M. Squyres, B. Barrett, and G .E. Fagg. The open run-time environment (openrte): A transparent multi-cluster environment for high-performance computing. In *Proceedings, 12th European PVM/MPI Users' Group Meeting*, Sorrento, Italy, September 2005.

[8] G. E. Fagg, A. Bukovsky, and J. J. Dongarra. HARNESS and fault tolerant MPI. *Parallel Computing*, 27:1479–1496, 2001.

[9] 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 Fe, NM, October 27-29 2003.

[10] E. Garbriel et al. Open MPI: Goals, concept, and design of a next generation MPI implementation. In *Proceedings, 11th European PVM/MPI Users' Group Meeting*, 2004.

[11] 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.

[12] 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.

[13] William Gropp, Steven Huss-Lederman, Andrew Lumsdaine, Ewing Lusk, Bill Nitzberg, William Saphir, and Marc Snir. *MPI — The Complete Reference: Volume 2, the MPI-2 Extensions*. MIT Press, 1998.

[14] Timothy G. Mattson, David Scott, and S.R.W. A teraflops supercomputer in 1996: The asci tflop system. In *Proceedings of the 1996 International Parallel Processing Symposium*, 1996.

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

[16] Rolf Riesen Ron Brightwell, Arthur B. Maccabe. Design, implementation, and performance of mpi on portals 3.0. *International Journal of High Performance Computing Applications*, 17(1), Spring 2003.

[17] L. Shuler, C. Jong, R. Riesen, D. van Dresser, A.B. Maccabe, L.A. Fisk, and T.M. Stallcup. The puma operating system for massively parallel computers. In *Proceedings of the 1995 Intel Supercomputer User's Group Conference*, 1995.

[18] Marc Snir, Steve W. Otto, Steve Huss-Lederman, David W. Walker, and Jack Dongarra. *MPI: The Complete Reference*. MIT Press, Cambridge, MA, 1996.

[19] 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.

[20] J.M. Squyres and A. Lumsdaine. A Component Architecture for LAM/MPI. In *Proceedings, 10th European PVM/MPI Users' Group Meeting*, Lecture Notes in Computer Science, Venice, Italy, September 2003. Springer-Verlag.

[21] T.S. Woodall et al. Open MPI's TEG point-to-point communications methodology : Comparison to existing implementations. In *Proceedings, 11th European PVM/MPI Users' Group Meeting*, 2004.

[22] T.S. Woodall et al. TEG: A high-performance, scalable, multi-network point-to-point communications methodology. In *Proceedings, 11th European PVM/MPI Users' Group Meeting*, Budapest, Hungary, September 2004.