Designing a Portable MPI-2 over Modern Interconnects Using uDAPL Interface

L. Chai, R. Noronha, P. Gupta, G. Brown, and D. K. Panda

Department of Computer Science and Engineering, The Ohio State University, USA
{chail, noronha, guptapr, browngre, panda}@cse.ohio-state.edu

Abstract. In the high performance computing arena, there exist several implementations of MPI-1 and MPI-2 for different networks. These implementations are highly optimized for their native networks. Some implementations allow the developer to work with multiple networks. However, most of them require the implementation of a new device, before they can be deployed on a new networking interconnect. The emerging uDAPL interface provides a network-independent interface to the native transport of different networks. Designing a portable MPI library with uDAPL might allow the user to move quickly from one networking technology to another. In this paper, we have designed the popular MVAPICH2 library to use uDAPL for communication operations. To the best of our knowledge, this is the first open-source MPI-2 compliant implementation over uDAPL. Evaluation with micro-benchmarks and applications on InfiniBand shows that the implementation with uDAPL performs comparably with that of MVAPICH2. Evaluation with micro-benchmarks on Myrinet and Gigabit Ethernet shows that the implementation with uDAPL delivers performance close to that of the underlying uDAPL provider.

Keywords: MPI-1, MPI-2, uDAPL, InfiniBand, Myrinet, Gigabit Ethernet, Cluster

1 Introduction

Message Passing is a popular paradigm for writing parallel applications. Application written with Message Passing Interface (MPI) libraries like MPICH and MPICH2 [3] from Argonne National Labs may be run on a wide variety of architectures and networks. This is easily achieved by relinking the application with the appropriate library for that architecture or network.

Modern interconnection technologies like InfiniBand [7], 10 GigE, Myrinet [14] and Quadrics [17] offer improved performance. This is both in terms of lower latency of the order of a few micro-seconds and higher bandwidth. Moreover, most of these networks allow low overhead, kernel-bypass user-level communication. This allows the user to maximize the communication performance achievable from these networks. Applications may also take advantage of the RDMA capabilities of these networks to read and write data with low overhead from each other's memory.

* This research is supported in part by Department of Energy's Grant #DE-FC02-01ER25506, and National Science Foundation's grants #CCR-0204429 and #CCR-0311542, and Grants from Mellanox, Intel and Sum MicroSystems

Though most high-speed networks offer user-level communication interfaces that are similar in semantics, they usually differ syntactically from each other. To avoid being tied down to a particular network, most application writers use a communication library abstraction such as MPICH. This approach allows the writer to exploit highly optimized libraries such as MVAPICH [15], MPICH-GM [12] and MPI/Elan4 [17]. While this approach works well, it might cause delays when a new networking technology comes out. The MPI library needs to be implemented on that network before the application can be moved.

One technique, to reduce the delay in moving to a new network, is to employ TCP as the communication protocol in MPICH. While this is a relatively easy option, the overhead of the TCP stack offsets the benefit of using a modern high performance network. The other technique requires the development of a new device for that particular network. Alternatively, one may use lower abstraction like the RDMA channel in MPICH2 to quicken the pace of the port. Though this may reduce the time for development, the stability and correctness of the implementation may still be an issue.

The emerging User Direct Access Programming Library (uDAPL) defines a standard and device independent interface for accessing the transport mechanisms of RDMA capable networks. This allows developers to design applications in a network independent fashion. Depending on the design of the underlying uDAPL, the performance impact of using the underlying uDAPL might be negligible.

With the development of uDAPL providers for a variety of networks, it is natural to ask whether a MPI library can be designed with an uDAPL interface. This might potentially have advantages on many levels. It would allow an application using an MPI library designed with an uDAPL transport to move seamlessly from one network to another. In addition, an extensively deployed and tested library may inspire more confidence than a newer implementation. Finally, depending on the performance of the underlying uDAPL provider library, there may be little degradation in performance compared with an application designed with a native interface. Thus, the open challenges are: 1. How to design such an MPI-2 library over the uDAPL interface, and 2. How much performance degradation this approach will lead to compared with an MPI-2 implementation over the native interface.

In this paper, we take on these challenges and design a portable MPI-2 with uDAPL and evaluate the performance trade-offs. We first present some background information on high-performance networks, uDAPL and MPI in section 2. Following that in section 3, challenges and design alternatives are presented. In section 4, the performance in terms of micro-benchmarks and applications is presented. Finally we conclude in section 5.

2 Background

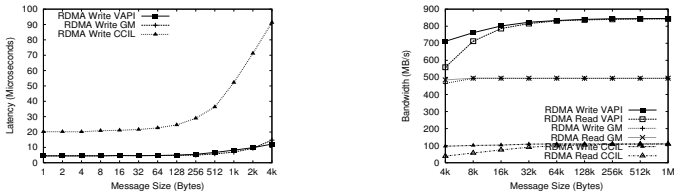
2.1 Overview of Interconnects

In the area of high performance computing, InfiniBand and Myrinet are popular and established technologies. These technologies support Remote Direct Memory Access (RDMA) [18]. The Ammasso 1100 is a relatively new entry. It is a RDMA-enabled Gigabit Ethernet adapter [2].

Figure 1 shows the native performance of these networks. The InfiniBand Architecture (IBA) [7] defines a System Area Network with a switched, channel-based interconnection fabric. IBA 4X has bandwidth up to 10 Gbps. The Myrinet [14] network technology utilizes programmable network interface cards (NIC) and

cut-through crossbar switches with operating system bypass techniques for full-duplex 4Gbps data rates. The Gigabit Ethernet used is a full-duplex 1Gbps Ethernet adapter that also supports RDMA.

The native programming interfaces of the networking technologies are VAPI, GM [13], and ccil [1] for IBA, Myrinet, and Gigabit Ethernet, respectively. OpenIB [16] is proposed as the next-generation software stack over InfiniBand, which is currently under development. In our experiments we still use VAPI.



(a) **Small Message Latency**

(b) **Large Message Bandwidth**

Fig. 1. Performance of InfiniBand, Myrinet, and Gigabit Ethernet

2.2 uDAPL

User Direct Access Programming Library (uDAPL) is a lightweight, transport-independent, platform-independent user-level library that provides a common API for all RDMA-enabled modern interconnects. uDAPL is defined by DAT (Direct Access Transport) Collaborative [5].

2.3 Message Passing Interface

MPI (Message Passing Interface) [20] is a standard library specification for message passing in parallel applications. MPI-2 [11] is an extension to MPI-1 standard. MPI-2 supports several new types of functionalities, including one-sided communication. Unlike send-receive model which is also referred to as two-sided communication, one-sided communication requires only one process to specify all communication parameters, and the other process is not involved in the communication. The two processes need explicit synchronization. We focus on active one-sided communication defined by MPI-2, which includes three functions: *MPI_Put*, *MPI_Get*, and *MPI_Accumulate*.

MVAPICH2 [15] is a high-performance MPI-2 implementation over InfiniBand. It is an implementation of MPICH2's [3] RDMA layer. MVAPICH2 is implemented on top of VAPI. The detailed design is discussed in [10, 9]. MVAPICH2 together with MVAPICH is currently being used by more than 200 organizations worldwide [15] to extract the benefits of InfiniBand for MPI applications. Commercial implementations of MPI with uDAPL include SCALI MPI [19] and Intel MPI [8]. Open MPI [6] enables the application to run across multiple networks, but (to the best of our knowledge) currently does not run over a uDAPL provider.

3 Design Issues

In this section we discuss the challenges related to designing and implementing a portable and efficient MPI-2 over uDAPL interface, and how we solve them. Our design is adapted from MVAPICH2.

As can be seen from Figure 2 MVAPICH2 has four major components: connection management, communication, memory management,

and descriptor management. In the mean time, uDAPL provides many features. Our goal is to design an adaptation layer to allow MVAPICH2 to run smoothly over uDAPL interface for both two-sided and one-sided communication.

With respect to communication and memory management components, we inherit the design from MVAPICH2. In this section we discuss in detail about connection and descriptor management. We also discuss about design issues in one-sided communication.

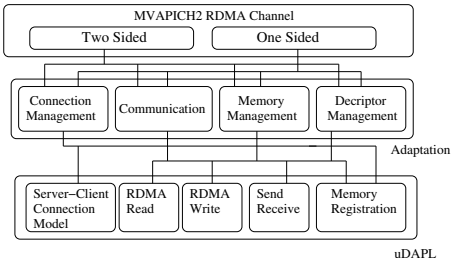
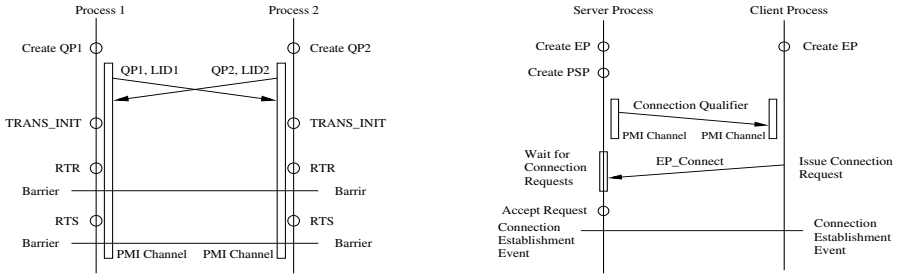


Fig. 2. Design 0 for MVAPICH2 communication interfaces with uDAPL

3.1 Connection Management

MPI communication assumes a fully connected topology. Since MVAPICH2 is based on Reliable Connection (RC) service of InfiniBand, every process needs to establish a connection with every other process in the initialization phase.



(a) **VAPI** (b) **uDAPL**

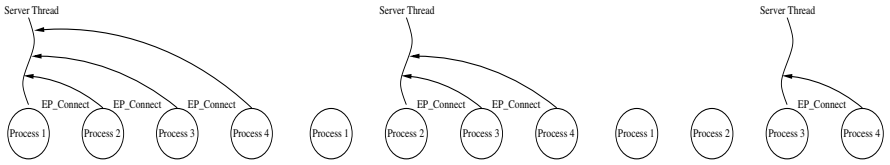
Fig. 3. Connection Establishment Models in VAPI and uDAPL

Figure 3(a) shows the InfiniBand/VAPI model of establishing a connection between two processes. The detailed discussion is in [7]. Both processes first create *Queue Pairs* (QPs), and then exchange QPs and *Local IDentifiers* (LIDs) through the *Process Management Interface* (PMI) [3]. Then both processes initialize QPs and transit to *Ready-To-Receive* (RTR) state. After a barrier through the PMI, both processes transit to *Ready-To-Send* (RTS) state. And after another barrier through the PMI, a connection is established between these two processes. uDAPL provides a server-client model for connection establishment which is totally different from InfiniBand/VAPI's peer-to-peer model, as shown in Figure 3(b). Both processes first create communication *Endpoints* (EPs). Then the server process creates a *Public Service Point* (PSP) which is a listen handle. Each PSP is associated with a system-wide unique *Connection Qualifier*. The server then gives the *Connection Qualifier* to the client through the PMI. After that the server listens on the PSP handle to wait for connection requests. The client issues a connection request (*EP_Connect*) to the PSP. Once the request is accepted by the server, a connection is established between the server and the client.

In order to achieve efficient connection establishment, we need to consider two issues. One is to avoid retransmission. When a client tries to connect to a server,

the server should already be listening there, otherwise the connection request will get rejected. The other issue is performance. Since every two processes need to establish a connection, we want this whole process to take place concurrently.

Having these two issues in mind, we propose the following approach, as shown in Figure 4. Every process is the server of the processes who have a higher global rank, and client of the processes who have a lower global rank. This means a process must listen persistently on the PSP handle while actively issue connection requests. In order to achieve this, we use a separate thread for server functionality. Every process first spawns a server thread, then after a synchronization, processes can request connections to their corresponding servers. The server thread exits once it has accepted the correct number of requests so that it will not affect any communication performance later. Using this approach, we can establish connections between every two processes in a reliable and efficient manner.



(a) **Process 1 is the server of Process 2, 3, and 4**

(b) **Process 2 is the server of Process 3 and 4**

(c) **Process 3 is the server of Process 4**

Fig. 4. Proposed Thread-based Connection Establishment Scheme. Actions shown in Figure (a), (b), and (c) take place concurrently.

3.2 Descriptor Management

In this section we discuss issues related to descriptor management. Both MVAPICH2 and MVAPICH2 with uDAPL use RDMA Write for eager protocol and RDMA Read for rendezvous protocol. When a RDMA operation is posted, information such as local address, segment length, remote address, etc. is encapsulated in a descriptor, and the descriptor is passed as an argument to the underlying VAPI or uDAPL functions. However, VAPI and uDAPL have different requirements for descriptor management.

There is no explicit descriptor management in MVAPICH2. Once a RDMA operation is posted, the descriptor is internally copied by VAPI. But for designing MVAPICH2 with uDAPL, we must carefully manage the descriptors because according to the uDAPL specification descriptors should not be modified until the corresponding RDMA operation has finished.

For using InfiniBand, buffers used for communication must be registered with InfiniBand Host Channel Adaptor (HCA). Since the registration process is time consuming, MVAPICH2 uses a set of pre-registered buffers for eager protocol. This allows us to use a simple and efficient method for descriptor management. We associate each buffer with a descriptor. Whenever a buffer can be reused - which means the previous RDMA Write associated with this buffer has finished - the corresponding descriptor can be safely reused. For rendezvous protocol, the communication buffer is registered on the fly. A descriptor is dynamically allocated for each buffer. The address of the descriptor is saved as a cookie. An uDAPL cookie is a user-supplied identifier for a *Data Transfer Operation* (DTO), which allows a user to uniquely identify the DTO when it completes. The cookie

is passed as an argument to the uDAPL post-RDMA-Read function. When a RDMA Read completes, we can use the cookie to retrieve the corresponding descriptor and free it. This descriptor management approach adds almost no overhead to the overall performance.

3.3 Design issues in One-sided Communication

The design issues discussed above also exist in one-sided communication. For connection management, we setup a separate set of EPs for one-sided communication, and connections are established in a similar manner as described in section 3.1. For descriptor management, since buffers are also pre-registered for eager protocol, and registered on the fly for rendezvous protocol, we use the same descriptor management scheme as described in section 3.2.

4 Performance Evaluation

In this section, we evaluate the performance of the MVAPICH2 with uDAPL. We first look at the experimental setup. Following that, MVAPICH2 with uDAPL is evaluated in terms of micro-benchmarks and applications over InfiniBand. Finally, experimental results for Myrinet and Ammasso Gigabit Ethernet are presented.

4.1 Experimental Setup

Two clusters are used for the evaluation. Cluster A has 8 nodes. Each node is equipped with dual 3.0 GHz processors, 512 KB L2 cache, 2GB main memory and 64-bit 133 MHz PCI-X interfaces. It is connected through MT23108 HCA's to a MT2400 switch. The uDAPL library from IBGD 1.6.1 is used. Each node also has a Myrinet E-card connected to a Myrinet-2000 switch. The uDAPL library 0.94+2 with GM 2.1.9 is used. Ammasso Gigabit Ethernet cards are also present on each node. They are connected to a Foundry switch. The uDAPL library provided by Ammasso Inc. is used.

Cluster B has 32 nodes. Each node has dual 2.6 GHz processors, 2GB main memory, 512KB L2 cache and 133 MHz PCI-X interfaces. Each node is connected to a MTS14400 switch with MT23108 HCA's. Mellanox IBGD-1.6.1 is used on all nodes.

4.2 Performance Evaluation over InfiniBand

Micro-benchmark Level Evaluation In this section, we evaluate the performance of MVAPICH2/uDAPL over InfiniBand using micro-benchmarks. The experiments are conducted on Cluster A as described in section 4.1.

Figure 5 shows the uDAPL-level and MPI-level ping-pong latency results. Small message latency of MVAPICH2/uDAPL/VAPI is around 6.7μs. It is 1μs higher than the latency of MVAPICH2/VAPI for messages smaller than 256 bytes.This is because MVAPICH2 utilizes *inline data transfer* scheme, where small data is sent in one message with the request, while uDPAL/VAPI doesn't utilize *inline data transfer*. For messages larger than 256 bytes, MVAPICH2/uDAPL and MVAPICH2 have comparable latency performance. The latency of MVAPICH2/uDAPL/VAPI is also close to that of uDAPL/VAPI for messages smaller than 256 bytes. After 256 bytes, the latency of both MVAPICH2 and MVAPICH2/uDAPL/VAPI goes up, because MPI-level messages need to be copied from user buffers to pre-registered RDMA buffers, and copy time goes up as message size grows.

From Figure 6 we can see that MVAPICH2/uDAPL/VAPI achieves the same bandwidth capability compared with MVAPICH2. The peak bandwidth is around 867 MillionBytes/second (MB/s).

Figure 7 shows the latency comparison of active one-sided communication. The small message latency of MVAPICH2/uDAPL/VAPI is 8.9μs for MPI_put,

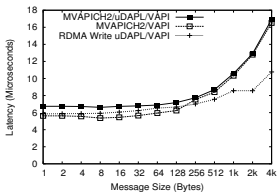


Fig. 5. Small Message Latency Comparison

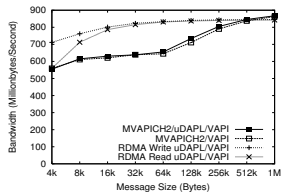
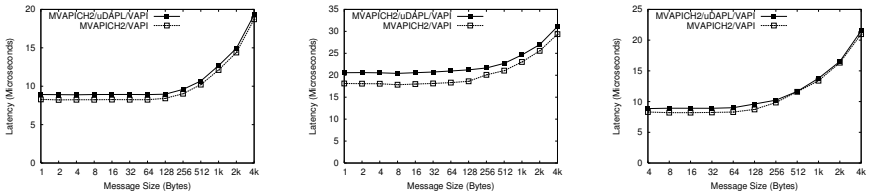


Fig. 6. Large Message Bandwidth Comparison



(a) **Put** (b) **Get** (c) **Accumulate**

Fig. 7. Latency Comparison of Active One-sided Communication

20.5μs for MPI_get, and 8.9μs for MPI_accumulate. MPI_put and MPI_get latency is about 8% higher than that of MVAPICH2/VAPI, and MPI_accumulate latency is about 6% higher. This can be affiliated to the two small synchronization messages which are not inlined.

Application Level Evaluation In this section, we evaluate the implementation of MVAPICH2/uDAPL/VAPI against MVAPICH2 using some of the NAS Parallel [4] and ASCI Blue [21] benchmarks. To evaluate MVAPICH2/uDAPL and MVAPICH2, the class A size of the applications FT, CG, LU and MG and class C size of IS were used. These are denoted by FT(A), CG(A), LU(A), MG(A) and IS(C) respectively. For the application Sweep3D, the large size 150 was used. This is denoted by S3D(150). All runs were with 64 processes on 32 nodes (Cluster B).

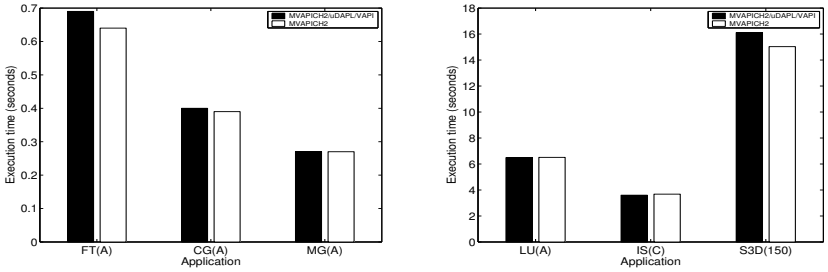
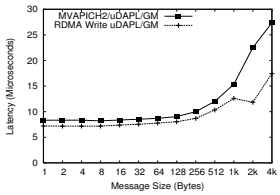
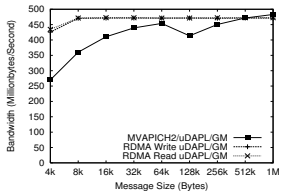


Fig. 8. Execution time of different applications with InfiniBand at 64 processes on 32 nodes using MVAPICH2/uDAPL and MVAPICH2.

Figure 8 shows the execution time of the applications. For FT(A), CG(A) and S3D(150), MVAPICH2/uDAPL performs slightly worse than MVAPICH2 by approximately 6%, 2.5% and 7% respectively. FT(A), CG(A) and S3D(150) use small messages in the range 0-256 bytes. As discussed in section 4.2, uDAPL/VAPI has approximately 1μ higher latency for the small message range of 0-256 bytes compared with the native VAPI. This difference in latency is reflected in the MVAPICH2/uDAPL implementation. For the other applications, MG(A), LU(A) and IS(C), MVAPICH2/uDAPL performs comparably with MVAPICH2.

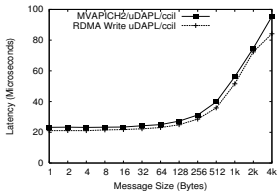


(a) **Small Message Latency**

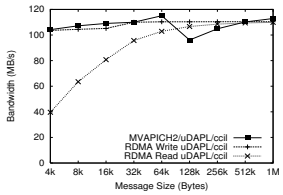


(b) **Large Message Bandwidth**

Fig. 9. Latency and Bandwidth Performance on Myrinet



(a) **Small Message Latency**



(b) **Large Message Bandwidth**

Fig. 10. Latency and Bandwidth Performance on Gigabit Ethernet

4.3 Performance Evaluation on Myrinet and Gigabit Ethernet

As can be seen from Figure 9, MVAPICH2/uDAPL/GM and uDAPL/GM have comparable latency for messages smaller than 1KB, around 8.3μs. After 1KB, copy overhead makes MVAPICH2/uDAPL/GM latency go higher. With respect to bandwidth, MVAPICH2/uDAPL/GM achieves the same peak bandwidth with uDAPL/GM, which is around 480MB/s.

From Figure 10(a), we can see the MVAPICH2/uDAPL/ccil latency performance is relatively close to uDAPL/ccil RDMA Write latency. Small message latency is around 21μs. The bandwidth of MVAPICH2/uDAPL/ccil also closely matches that of the uDAPL/ccil RDMA bandwidth. Peak bandwidth is about 110 MB/s.

5 Conclusions and Future Work

In this paper, we have designed a high-performance implementation of MVAPICH2 with uDAPL. The performance evaluation has been done on three different interconnects using both micro-benchmarks and applications. For InfiniBand, the implementation of MVAPICH2 with uDAPL performs comparably with that of MVAPICH2 on micro-benchmarks as well as applications. For Myrinet and Gigabit Ethernet (Ammasso TCP offload), the MVAPICH2 with uDAPL performs comparably with the uDAPL layer in terms of micro-benchmarks.

In the current implementation, uDAPL is used only on the point-to-point path. We plan on investigating how to design MPI collective operations using uDAPL. Additionally, we plan on studying the impact of moving the MPI-2 design from the RDMA channel to the ADI3 layer. *Software Distribution: The proposed MVAPICH2 with uDAPL interface will be released as an open-source version soon. Interested parties can visit [15] to download it.*

References

1. Ammasso, Inc. ccil API Documentation, December 2004.
2. Ammasso, Inc. The Ammasso 1100 High Performance Ethernet Adapter User Guide. http://www.ammasso.com/amso1100_usersguide.pdf, February 2005.
3. Argonne National Laboratory. MPICH - A Portable Implementation of MPI. http://www-unix.mcs.anl.gov/mpi/mpich.
4. D. H. Bailey, E. Barszcz, L. Dagum, and H.D. Simon. NAS Parallel Benchmark Results. Technical Report 94-006, RNR, 1994.
5. DAT Collaborative. http://www.datcollaborative.org/.
6. Edgar Gabriel and et al Graham E. Fagg. Open MPI: Goals, concept, and design of a next generation MPI implementation. In *Proceedings, 11th European PVM/MPI Users' Group Meeting*, Budapest, Hungary, September 2004.
7. Infiniband Trade Association. http://www.infinibandta.org/.
8. Intel Corporation. Intel MPI Libary for Linux v1.0.1 Release Notes. http://www.intel.com/software/products/cluster/mpi/relnotes.pdf, 2003-2005.
9. W. Jiang, J. Liu, H. Jin, D. K. Panda, W. Gropp, and R. Thakur. High performance mpi-2 one-sided communication over infiniband. In *IEEE/ACM International Symposium on Cluster Computing and the Grid*, 2004.
10. J. Liu, W. Jiang, Pete Wyckoff, D. K. Panda, D. Ashton, D. Buntinas, W. Gropp, and B. Toonen. Design and implementation of mpich2 over infiniband with rdma support. In *Int'l Parallel and Distributed Processing Symposium*, 2004.
11. Message Passing Interface Forum. MPI-2: A Message Passing Interface Standard. *High Performance Computing Applications*, 12(1–2):1–299, 1998.
12. MPICH-GM Software. *www.myrinet.com/scs*.
13. Myricom Inc. Myricom GM myrinet software and documentation. http://www.myri.com/scs/GM/doc/gm_toc.html, 2000.
14. N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C.L. Seitz, J. Seizovic, and W. Su. Myrinet - a gigabit per second local area network., February 1995.
15. Network-Based Computing Laboratory. MPI over InfiniBand Project. http://nowlab.cis.ohio-state.edu/projects/mpi-iba/index.html.
16. OpenIB.org. http://www.openib.org/.
17. Quadrics Ltd. *www.quadrics.com*.
18. RDMA Consortium. RDMA Protocol Verb Specification. http://www.rdmaconsortium.com/home, April 2003.
19. Scali, Inc. Single interconnect independent MPI across high performance clusters. http://www.scali.com/index.php?loc=18, 2004.
20. Marc Snir, Steve Otto, Steve Huss-Lederman, David Walker, and Jack Dongarra. *MPI–The Complete Reference. Volume 1 - The MPI-1 Core, 2nd edition.* The MIT Press, 1998.
21. The ASCI Blue Benchmarks. http://www.llnl.gov/asci_benchmarks.