MPI over InfiniBand: Early Experiences

Jiuxing Liu, Jiesheng Wu, Sushmitha P. Kini, Ranjit M. Noronha, Pete Wyckoff† and Dhabaleswar K. Panda

Network-Based Computing Laboratory Computer and Information Science
The Ohio State University
Columbus, OH 43210
{liuj, wuj, kinis, noronha, panda}@cis.ohio-state.edu

†Ohio Supercomputer Center
1224 Kinnear Road
Columbus, OH 43212
pw@osc.edu

December 23, 2002

OSU-CISRC-10/02-TR25

Visit nowlab.cis.ohio-state.edu/projects/mpi-iba/ for additional information and download of MVAPICH software.
Contact Prof. D. K. Panda (panda@cis.ohio-state.edu) for more details.

MPI over InfiniBand: Early Experiences*

Jiuxing Liu, Jiesheng Wu, Sushmitha P. Kini, Ranjit M. Noronha, Pete Wyckoff† and Dhabaleswar K. Panda

Computer and Information Science
The Ohio State University
Columbus, OH 43210
{liuj, wuj, kinis, noronha, panda}@cis.ohio-state.edu

†Ohio Supercomputer Center
1224 Kinnear Road
Columbus, OH 43212
pw@osc.edu

December 23, 2002

Abstract

Recently, InfiniBand Architecture (IBA) has been proposed as the next generation interconnect for I/O and inter-process communication. The main idea behind this industry standard is to use a scalable switched fabric to design the next generation clusters and servers with high performance and scalability. This architecture provides a plethora of new mechanisms and services (such as multiple transport services, RDMA and atomic operations, multicast support, service levels and virtual channels). This raises an interesting challenge about how to take advantage of these features to design scalable communication subsystem for next generation clusters. As the second generation IBA products are rolling out, this paper presents our early experience in designing and implementing Message Passing Interface (MPI) on top of the Verbs layer of InfiniBand. Challenges in designing such an implementation are outlined. We evaluated our MPI implementation with Mellanox's second generation adapter (InfiniHost) and switch (InfiniScale). On 2.4 GHz systems with PCI-X 133 MHz interfaces, our MPI implementation over Verbs layer with RDMA write support is shown to deliver around 11.6 microsec latency for short messages and a bandwidth of over 750 MBytes/sec for long messages. The Verbs layer on the InfiniHost adapter itself delivers 8.1 microsec latency for small messages and up to 820 MBytes/sec bandwidth for large messages. On the same testbed, a detailed performance evaluation (user-level communication, MPI-level, and application-level) is carried with other contemporary quoted interconnects such as Myrinet and Quadrics and their respective MPI implementations. The Mellanox's InfiniBand adapters and our MPI implementation are shown to deliver better performance for some cases compared to Myrinet and Quadrics. InfiniBand products are currently undergoing rapid growth and optimization. Thus, it is expected that the performance numbers for near future InfiniBand adapters will be even better. These results demonstrate that IBA can be used to build next generation large-scale clusters and data centers with high performance.

*This research is supported in part by Sandia National Laboratory's contract #30505, Department of Energy's Grant #DE-FC02-01ER25506, and National Science Foundation's grants #EIA-9986052 and #CCR-0204429.

1 Introduction

Emerging distributed and high performance applications require large computational power as well as low latency, high bandwidth and scalable communication subsystems for data exchange and synchronous operations. In the past few years, the computational power of desktop and server computers has been doubling every eighteen months. The raw bandwidth of network hardware has also increased to the order of Gigabit per second. However, the performance of underlying hardware is not delivered to the applications due to the traditional architectures of communication subsystems. The traditional networking protocols for inter-processor communication and I/O usually need heavy kernel involvement and extra data copies in the communication critical path. TCP/IP is the most popular one due to its flexibility and layered structure. However, it takes a lot of CPU power to provide reasonable communication performance. These factors lead to the degradation of communication performance as seen by the application layer.

During the past few years, the research and industry communities have been proposing and implementing many communication systems such as AM [25], VMMC [4], FM [19], U-Net [24, 26], LAPI [22], and BIP [21] to address some of the problems associated with the traditional networking protocols. In these systems, the involvement of operating system kernel is minimized and the number of data copies is reduced. As a result, they can provide much higher communication performance to the application layer. In the recent past, the Virtual Interface Architecture (VIA) [8, 6] was proposed to standardize these efforts.

More recently, InfiniBand Architecture [12] has been proposed as the next generation interconnect for I/O and inter-process communication. In InfiniBand, computing nodes and I/O nodes are connected to the switched fabric through Channel Adapters. InfiniBand provides a Verbs interface which is a superset of VIA. This interface is used by the host operating systems to communicate with Host Channel Adapters. InfiniBand provides many novel features: three different kinds of communication operations (send/receive, RDMA, and atomic), multiple transport services (such as reliable connection (RC), unreliable datagram (UD), and reliable datagram (RD), different mechanisms for QoS (such as service levels and virtual lanes). In addition to providing scalability and high performance, InfiniBand also aims to meet applications' need for Reliability, Availability and Serviceability (RAS).

Recently, several companies have started shipping out InfiniBand products (such as Host Channel Adapters and switches). The novel features associated with the InfiniBand and the availability of InfiniBand products lead to the following open question for computer architects: *How to design scalable and high performance communication subsystems for next generation clusters with the InfiniBand Architecture?*

In the parallel computing area, Message Passing Interface (MPI) [14, 23] has become the de-facto standard for developing portable parallel applications. However, to the best of our knowledge, there is no MPI implementation available which sits on top of the native Verbs layer of InfiniBand. There is also little insight regarding performance of InfiniBand network with other contemporary networks such as Myrinet [5] and Quadrics [20] at various levels: network-level, MPI-level, and applications-level.

In this paper, we take on this challenge and present our early experience in implementing MPI on InfiniBand's verbs layer. Specifically, we address the following issues in the paper:

1. Design challenges in implementing MPI layer over the verbs interface of InfiniBand architecture
2. Preliminary implementation of the MPI layer and its performance evaluation at the MPI-level and applications-level.
3. Performance comparison of InfiniBand network and its MPI implementation with other contemporary cluster interconnects (such Myrinet and Quadrics) and their MPI implementations.

Our design and evaluations are based on the second generation (4X links capable of driving up to 10.0 Gigabits/sec) InfiniBand adapters (InfiniHost) and 4X switch (InfiniScale) from Mellanox [1]. These adapters are connected to 2.4 GHz Xeon systems with PCI-X 133 MHz bus. Our MPI implementation is shown to deliver around 11.6 microsec latency for short messages and over 750 MBytes/sec bandwidth for large messages. The impact of different components of the MPI implementation are investigated in detail.

Performance evaluation at the Verbs-level API (VAPI) layer demonstrates that the second generation InfiniBand adapters (such as InfiniHost) are capable of delivering 8.1 microsec latency for small messages and up to 820 MBytes/sec bandwidth for large messages. Thus, our preliminary MPI implementation puts very little overhead on top of the VAPI layer.

A performance comparison of the VAPI layer with Myrinet/GM [18] and Quadrics/Elan [20] on the same 2.4 GHz systems [1] indicates that as message size increases, the VAPI layer on InfiniBand provides lower latency and higher bandwidth.

At the MPI-level performance comparison, as message size increases, MPI/VAPI over InfiniBand starts demonstrating better performance (lower latency and higher bandwidth) compared to MPICH/GM [16] over Myrinet. MPI over Quadrics shows better latency compared to MPI/VAPI layer over InfiniBand. However, MPI/VAPI layer delivers higher bandwidth (for larger messages) compared to MPI over Quadrics.

On a 4-node testbed, our results show that NAS Parallel Benchmarks run up to 35.7% faster on our MPI over InfiniBand than MPI over GM. We also find that our MPI implementation can outperform MPI over Quadrics by up to 18.6% for some NAS Parallel Benchmark applications.

Please note that the performance results presented here are preliminary and is based on early microcode release for InfiniHost. As the InfiniBand products are rapidly maturing, we expect to see upcoming firmware releases to continue reducing latency and increasing bandwidth at the VAPI layer. This will have direct impact on the MPI-level performance and applications-level performance. As new firmware releases will be available, we will generate the new performance results and make them available at the following URL: nowlab.cis.ohio-state.edu/projects/mpi-iba/ .

Currently our implementation runs on InfiniHost HCAs, which are second generation InfiniBand products from Mellanox. However, our implementation is based on the InfiniBand Verbs layer and it can be easily ported to other InfiniBand platforms.

The rest of the paper is organized as follows: In section 2, we provide a brief overview of the InfiniBand Architecture. In section 3, we introduce the hardware architecture of Mellanox InfiniHost cards. InfiniBand software architecture is presented in Section 4. Issues related to designing MPI layer over the Verbs Interface are presented in Section 5 and implementation details in Section 6. Performance evaluation results (user-level, MPI-level, and application-level) are presented and discussed in section 7. Finally, conclusions and future work are presented in section 8.

2 InfiniBand Architecture Overview

The InfiniBand Architecture defines a System Area Network (SAN) for interconnecting processing nodes and I/O nodes. Figure 1 provides an overview of the InfiniBand architecture. It provides the communication and management infrastructure for inter-processor communication and I/O. The main idea is to use a switched, channel-based interconnection fabric. The switched fabric of InfiniBand Architecture provides much more aggregate bandwidth. Also, a switched fabric can avoid single point of failure and provide more reliability. InfiniBand Architecture also has built-in QoS mechanisms which provide virtual lanes on each link and define service levels for each packets.

In an InfiniBand network, processing nodes and I/O nodes are connected to the fabric by Channel Adapters (CA). Channel Adapters usually have programmable DMA engines with protection features. They generate and consume IBA packets. There are two kinds of channel adapters: Host Channel Adapter (HCA) and Target Channel

[1] Please note that Myrinet LANai 9.0 and Quadrics Elan cards currently can work with only PCI-II 64x66 interface.

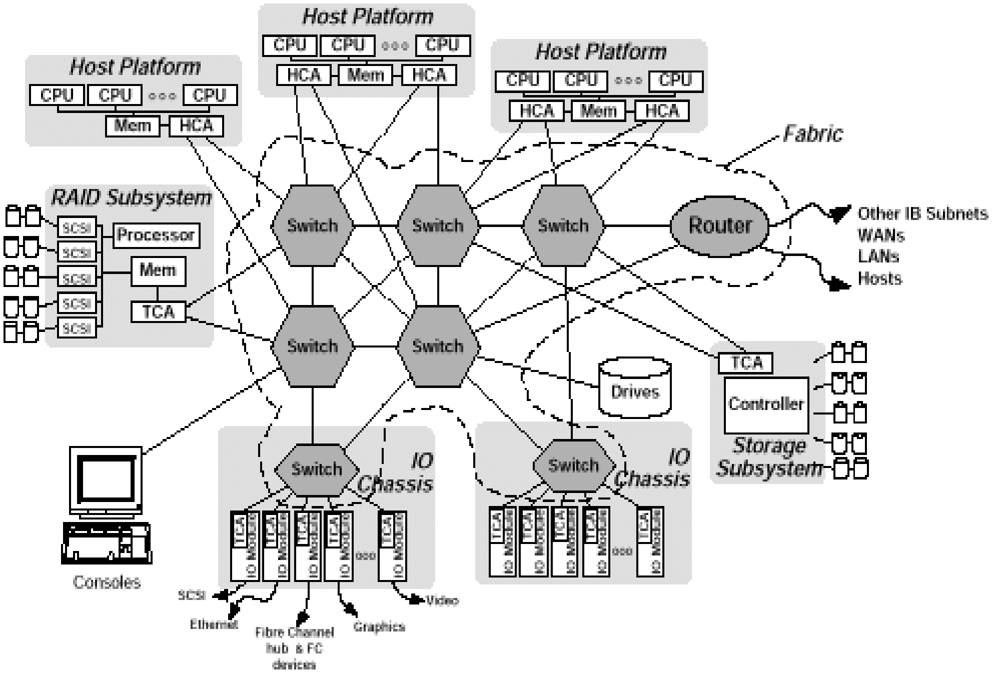

Figure 1. Illustrating a typical system configuration with the InfiniBand Architecture. (Courtesy InfiniBand Trade Association)

Adapter (TCA). HCAs sit on processing nodes. Their semantic interface to consumers is specified in the form of InfiniBand Verbs. Unlike traditional network interface cards, Host Channel Adapters are connected directly to the system controller. TCAs connect I/O nodes to the fabric. Their interface to consumers are usually implementation specific and thus not defined in the InfiniBand specification.

The InfiniBand communication stack consists of different layers. The interface presented by Channel adapters to consumers belongs to the transport layer. A queue-based model is used in this interface. A Queue Pair in InfiniBand Architecture consists of two queues: a send queue and a receive queue. The send queue holds instructions to transmit data and the receive queue holds instructions that describe where received data is to be placed. Communication operations are described in Work Queue Requests (WQR) and submitted to the work queue. Once submitted, a Work Queue Request becomes a Work Queue Element (WQE). WQEs are executed by Channel Adapters. The completion of work queue elements is reported through Completion Queues (CQ). Once a work queue element is finished, a completion queue entry is placed in the associated completion queue. Applications can check the completion queue to see if any work queue request has been finished. This structure is shown in Figure 2.

3 Mellanox Hardware Architecture

Our InfiniBand platform consists of InfiniHost HCAs and a InfiniScale switches from Mellanox [1]. In this section we will give a brief introduction to both the HCAs and the switch.

Infiniscale is a full wire-speed switch with eight 4X ports or 1X Infiniband Ports. These ports have an integrated 2.5 Gb/s physical layer serializer/deserializer and feature auto-negotiation between 1X and 4X links. The switch supports a 32 bit/66 Mhz PCI v2.2 compatible interface and a maximum transmission unit (MTU) of upto 4K. There is also support for eight Virtual Data Lanes (VLs) in additon to a Dedicated Management Lane (VL15).

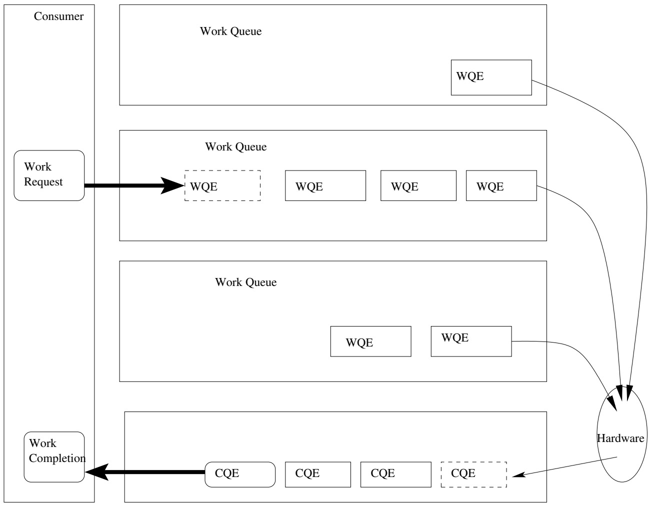

Figure 2. InfiniBand Work Queue and Completion Queue Structures

Addtionally, there is also support for link packet buffering, inbound and outbound partition checking and autonegotiation of link speed. Finally, the switch has an embedded RISC processor for exception handling, out of band data management support and performance monitoring counter support.

The Infinihost MT23108 dual 4X ported HCA/TCA allows for a bandwidth of up to 10 Gbit/s over its ports. It can potentially support upto 16 million QP's, EE's and CQ's. Memory protection along with address translation is implemented in hardware itself. PCI-X support along with DDR memory allows portions of host memory to be configured as a part of system memory using a tranparent PCI bridge allowing the host to directly place HCA related data without going over the PCI-X bus. The DDR memory allows the mapping of different queue entries namely work queues entries (WQE's) and execution queue entries to different portions of the system space transparently. At its heart, the HCA picks WQE's in a round robin fashion (the scheduler is flexible and supports more complex scheduling including weighted round robin with priority levels) and posts them to execution queues allowing for the implementation of QoS at a process level. Different WQE's specify how the completion notification should be generated. In the following section, we discuss the software interface to Infiniband.

4 InfiniBand Software Interface

Unlike other specifications such as VIA, InfiniBand Architecture doesn't specify an API. Instead, it defines the functionality provided by HCAs to operating systems in terms of Verbs, as shown in Figure 3. The Verbs interface specifies such functionality as transport resource management, multicast, work request processing and event handling.

Although in theory APIs for InfiniBand can be quite different from the Verbs interface, in reality many existing APIs have followed the Verbs semantics. One such example is the VAPI interface [1] from Mellanox. Many VAPI functions are directly mapped from corresponding Verbs functionality. This approach has several advantages: First, since the interface follows closely to the Verbs, the efforts needed to implement it on top of HCA is reduced. Second, because the Verbs interface is specified as a standard in the InfiniBand Architecture, it makes the job much easier to port applications from one InfiniBand API to another if they are both derived from Verbs.

As we have mentioned earlier, the communication in Verbs is based on queue-pairs. InfiniBand communication supports both channel (send/receive) and memory (RDMA) semantics. These operations are specified in work queue requests and posted to send or receive queues for execution. The completion of work queue requests is reported through completion queues (CQs). Although powerful and simple, the queue-pair based communication also has several restrictions: First, all communication memory must be registered first. This step is necessary

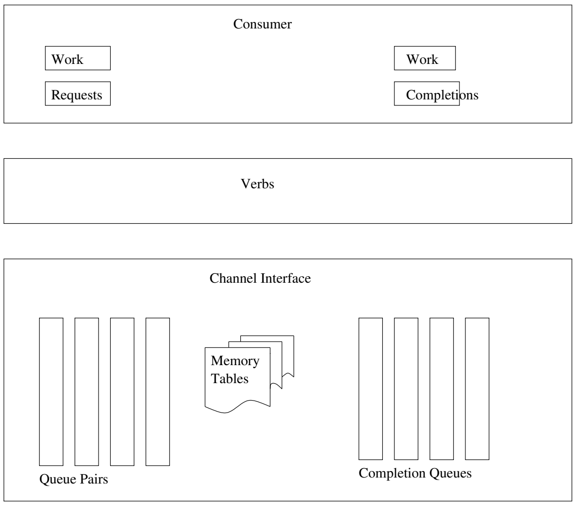

Figure 3. Verb Interface

because the HCA uses DMA operation to send from or receive into host communication buffers. These buffers must be pinned in memory and the HCA must have the necessary address information in order to carry out the DMA operation. The second restriction is that the sender must make sure a receive work queue request before it posts a send work queue request. Otherwise a retry mechanism will be triggered which may significantly reduce the performance. The receive also has to match the send in terms of buffer size. These issues must be addressed by software above the Verbs layer.

5 Putting MPI Layer over Verbs Interface

There are many design issues involved in putting an MPI layer on top of the Verbs Interface. InfiniBand provides two kinds of communication semantics: send/receive and RDMA. Although it is possible to implement MPI using only one of them, it is better to combine them so that we can take advantage of both semantics. MPI assumes that the underlying communication layer provides such functionality as reliability, buffer management, and flow control. MPI applications should not be involved in these issues. Since the Verbs Interface requires that communication buffers should be registered and upper layer software should take care of flow control, there is a functional mismatch between the MPI and the Verbs layer. Thus, the most important task for implementing MPI on top of the Verbs layer is to bridge this gap. InfiniBand provides different classes of services including reliable connection (RC) and reliable datagram (RD). If these two services are used for MPI, reliability can be guaranteed. However, buffer registration and flow control issues still need to be handled explicitly by the MPI implementation.

In addition to basic communication operations, InfiniBand also offers advanced features such as end-to-end flow control, atomic operations and QoS support. It is possible to improve MPI performance by taking advantage of these advanced features. However, many of these features are optional and they may not be available in every platform. Therefore, we must be careful using these features if we want to make our implementation portable. The current firmware release of InfiniHost adapters do not support those features. Thus we do not focus on these features in this paper.

5.1 Send/Receive vs RDMA

RDMA operations have the advantage that they are one sided and do not involve the remote process. Therefore they have less overhead. However, one restriction for using RDMA operations is that the destination address must be known in advance.

MPI implementations usually use two internal protocols to handle communication: Eager and Rendezvous. In Eager protocol, a message is sent to the receiver even though the corresponding receive has not been issued. In this case, the message is put into an unexpected queue and later copied to the receive buffer. In Rendezvous protocol, the actual data transfer takes place only after both send and receive have been issued.

Eager protocol matches well with send/receive operations in InfiniBand. For Rendezvous protocol, RDMA can also be used. This is because during Rendezvous protocol, the sender and the receiver exchange information through control messages before the actual data transfer, and the destination address needed by RDMA operation can be put into these control messages.

When to switch from one protocol to another depends on the messages size. Usually, small messages uses Eager protocol and large messages uses Rendezvous protocol. The switch point is important and it must be carefully chosen to match the performance characteristics of the underlying platform.

5.2 Handling Buffer Registration

Buffer registration and de-registration in InfiniBand Architecture are expensive operations, because they not only involve the operating system kernel, but also need some interaction between the NIC and the host. Therefore we would like to avoid these operations on the communication critical path if possible.

One way to address this problem is to maintain a pre-registered buffer pool. When the message is being sent, it is first copied to a buffer in the pool. Similarly, on the receiver side, messages are first received into buffers in the pre-registered pool, and then copied to the destination buffers. This method avoids the buffer registration overhead completely at the cost of two extra copies for very messages.

The other way is to use a technique called Pin-down Cache which is first proposed in [11]. The idea is to maintain a cache of registered buffers. When a buffer is first registered, it is put into the cache. When the buffer is unregistered, the actual unregister operation is not carried out and the buffer stays in the cache. Thus the next time when the buffer needs to be registered, we need not to do anything because it is already in the cache. A buffer is unregistered only when it is evicted from the cache. The effectiveness of Pin-down Cache depends on how often the application reuses its buffers. If the reuse rate is high, most of the buffer registration and de-registration operations can be avoided.

5.3 Flow Control

For send/receive operations in InfiniBand, when a message is sent out and there is no corresponding receive posted on the receiver side, a retry mechanism is triggered and the performance may drop significantly. In order to avoid buffer overrun on the receiver side, a flow control scheme is needed for the MPI implementation. Even when we use RDMA operations, flow control is still needed because the control messages may still use send/receive operations.

To deal with this problem, a credit-based flow control mechanism can be used. When a message is sent out, the sender's credit is decremented. When the receiver reposts receive requests, it can inform the sender that new credits are available. If the number of credits is low, the sender will switch from Eager protocol to Rendezvous. If there is no credit available, send operations will be blocked until enough credits arrive from the receiver. In this scheme, the number of credits may be important as it may affect the communication performance.

Another problem is that send and receive must match in terms of buffer size. A send operation cannot send a buffer which is larger than the corresponding receive buffer. Since the receiver may not know the buffer size in advance, messages usually are divided into chunks of fixed size. The receiver will post buffers for this particular size only. The size of buffer must be carefully chosen because if it is too large, most space in the buffers is wasted; if it is too small, the overhead of fragment messages and posting send/receive operations increases.

6 Implementation

Currently one of the most popular MPI implementations is MPICH [10] from Argonne National Laboratory. MPICH uses a layered approach in its design. The platform dependent part of MPICH is encapsulated by an interface called Abstract Device Interface, which allows MPICH to be ported to different communication architectures. Our MPI implementation on Mellanox InfiniHost cards is essentially an ADI2 (the second generation of Abstract Device Interface) implementation which uses VAPI as the underlying communication interface. Our implementation is also derived from MVICH [13], which is an ADI2 implementation for VIA. MVICH was developed in Lawrence Berkeley National Laboratory.

Currently our MPI implementation uses VAPI send/receive operations for Eager protocol. For Rendezvous protocol, RDMA write operations are used for data transfer while send/receive operations are used for control messages.

7 Performance Evaluation

In this section we present performance evaluation for our MPI implementation. Performance results at the VAPI level are first given. We also compare our results with those from Myrinet/GM and Quadrics, both at the MPI level and the underlying communication interface level. However, we need to point out that Myrinet and Quadrics cards use PCI-II 64x66 MHz interface while the InfiniHost HCAs use PCI-X 133 MHz interface. Therefore the performance number on these different platforms are not compared in entirely equivalent conditions.

7.1 Experimental setup

Our experimental testbed consists of a cluster system consisting of 4 SuperMicro SUPER P4DL6 nodes. Each node has dual Intel Xeon 2.40 GHz processors with a 512K L2 cache at a 400 MHz front side bus. The machines were connected by Mellanox InfiniHost MT23108 DualPort 4X HCA adapter through an InfiniScale MT43132 Eight 4x Port InfiniBand Switch. The HCA adapters work under the PCI-X 64-bit 133MHz interfaces. The machines were also connected by Myrinet network using NICs with 133MHz LANai 9 processors through a 8-port Myrinet-2000 switch. Myrinet adapters use 64-bit 66MHz PCI bus for all experiments. The Quadrics Elan3 QM-400 cards were attached to these fours nodes. They were connected with each other through a Elite16 switch. The QM-400 card also uses a 64-bit 66MHz PCI slot. We used the Linux Red Hat 7.1 operating system.

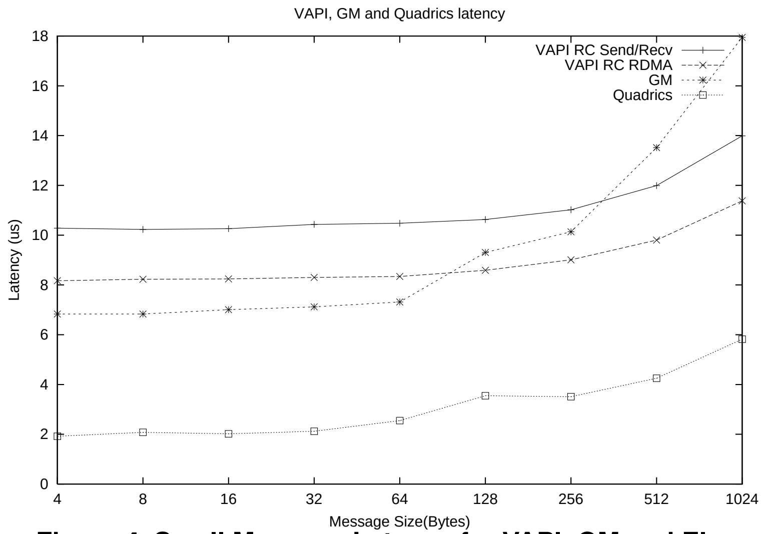

Figure 4. Small Message Latency for VAPI, GM and Elan

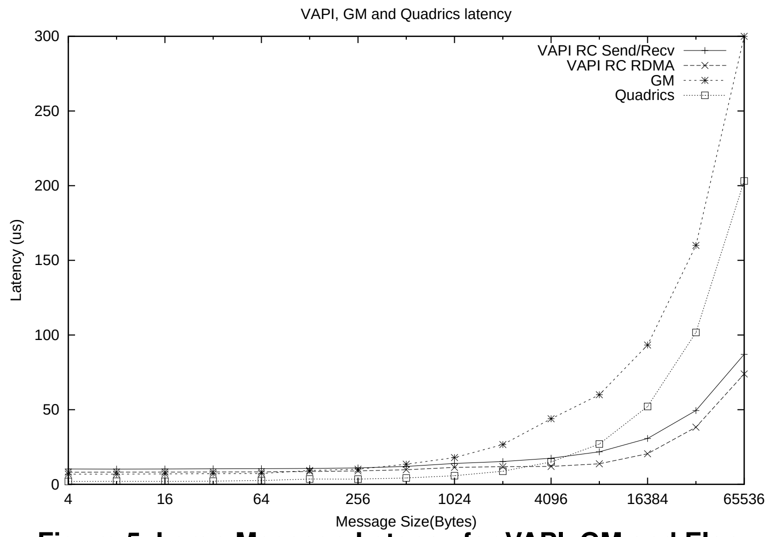

Figure 5. Large Message Latency for VAPI, GM and Elan

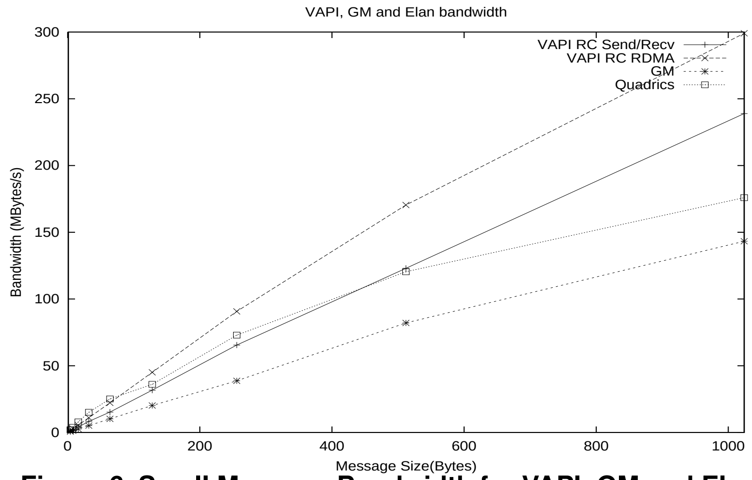

Figure 6. Small Message Bandwidth for VAPI, GM and Elan

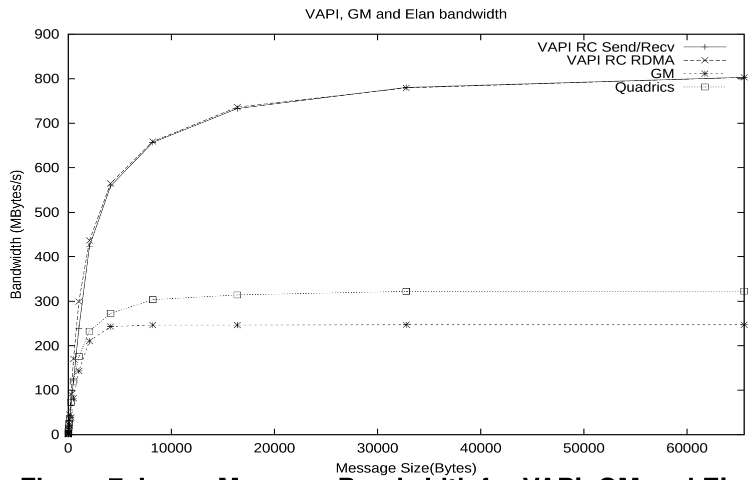

Figure 7. Large Message Bandwidth for VAPI, GM and Elan

7.2 VAPI Level Performance

The VAPI level latency results of InfiniHost cards are shown in Figures 4 and 12. The latency tests are carried out in a ping-pong fashion. In the bandwidth test, the sender keeps sending a certain number of messages to the receiver and then waits for a reply. For small messages, the latency for InfiniHost is around 10 microseconds for send/receive operation. The latency for RDMA write operation is around 8 microseconds and for GM it is about 7 microseconds. The latency results for InfiniHost is based on early microcode and we expect later firmware optimization will continue to reduce latency. Compared with InfiniHost VAPI, Myrinet/GM performs very good in latency for small messages (less than 256 bytes). But after that InfiniHost VAPI latency becomes smaller. InfiniHost VAPI performs significantly better compared to Myrinet/GM for large messages. Compared with InfiniHost VAPI, Quadrics Elan has better latency up to 4 KBytes. After that InfiniHost VAPI has better latency.

Figures 6 and 13 show the bandwidth results. The peak bandwidth for InfiniHost is around 820 MBytes/sec and for GM it is around 240 MBytes/sec. We have observed that the bandwidth of InfiniHost VAPI is better than Myrinet/GM in almost all message ranges. Compared with Quadrics Elan, InfiniHost VAPI performs better except for very small messages.

7.3 MPI Level Performance

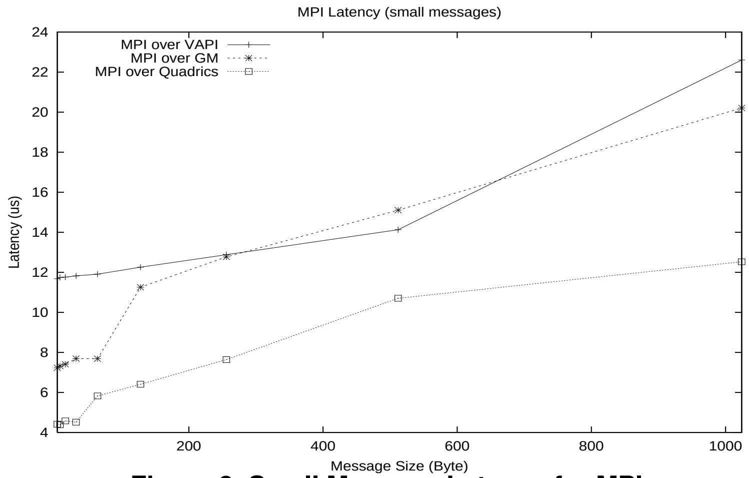

Figure 8. Small Message Latency for MPI

Figures 8 and 14 show the latency for different MPI implementations. The latency and bandwidth tests at the MPI level are similar to those at the VAPI. The smallest latency we have achieved is around 11.6 microseconds. Comparing our implementation with MPI over GM, we find that the two performs comparably for messages up to 4 KBytes. (GM performs better for messages smaller than 256 bytes.) However, for large messages MPI over VAPI performs better. MPI over Quadrics performs best for message range 0 to 16 KBytes and its smallest latency is around 4.3 microseconds.

The bandwidth graphs in Figure 10 and 15 show that our MPI implementation is able to achieve over 750 MBytes/sec peak bandwidth using RDMA. MPI over GM performs comparably with MPI over VAPI in the message range from 1 KBytes to 8 KBytes. For all other message sizes, MPI over VAPI gives better performance. Compared with MPI over Quadrics, our MPI implementation has better bandwidth except for messages around 5 KBytes where we switch from Eager protocol to Rendezvous protocol.

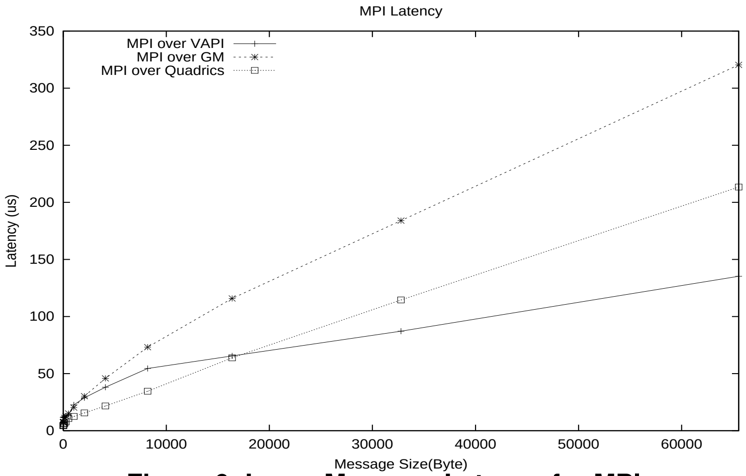

Figure 9. Large Message Latency for MPI

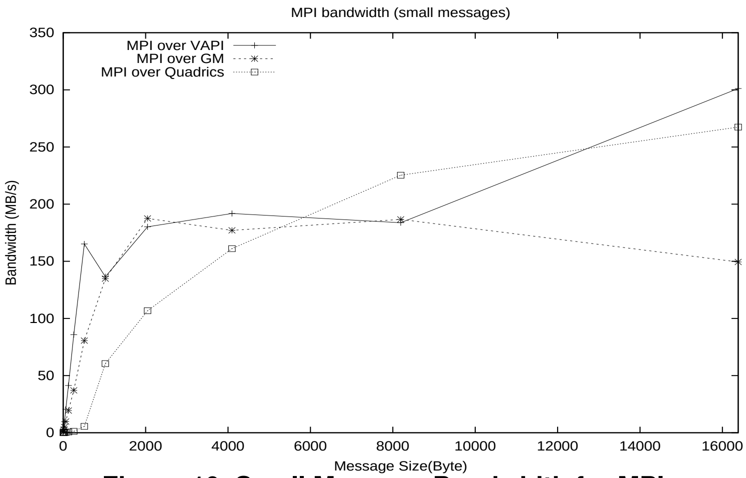

Figure 10. Small Message Bandwidth for MPI

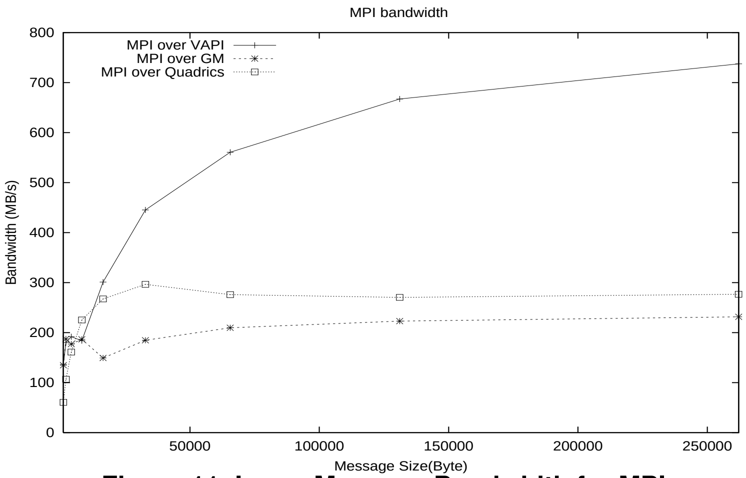

Figure 11. Large Message Bandwidth for MPI

7.4 Application Level Performance

7.4.1 NAS Parallel Benchmark Results

In Table 1 and Table 4 we show the running time of the NAS Parallel Benchmark suite [17]. Table 1 shows the results we got with GNU compilers and Table 4 shows the results with Intel compilers. (We were not able to run SP on MPI over GM with GNU compiler. And FT doesn't compile using GNU compiler.) The GNU compilers we were using are GNU GCC 2.96 and GNU FORTRAN 0.5.26. For Intel compilers, we were using Intel(R) C++ and FORTRAN Compilers for 32-bit applications Version 6.0.1 Build 20020822Z.

From the tables we can see that MPI over VAPI performs better than MPI over GM. For IS, MPI over VAPI has a 29.3 percent improvement in terms of running time using GNU compiler, and the improvement is 35.7 percent using Intel compiler. This improvement is largely due to the superior large message bandwidth for MPI over VAPI. For all NAS applications, on the average MPI over VAPI performs 12.1 percent better using GNU compilers and 11.6 percent better using Intel compiler.

Compared with MPI over Quadrics, MPI over VAPI gives better performance for 5 of the 6 NAS Benchmarks using GNU compilers. If we use Intel compiler, MPI over VAPI performs better for 4 of the 7 NAS Benchmarks. For GNU compilers, the best improvement we got is 12.8 percent from LU. For Intel compilers, IS gives about 18.6 percent improvement, which is the largest among all applications.

Table 1. NAS on MPI (4 Nodes, Class A, GNU compiler)

	VAPI(s)	GM (s)	Quadrics (s)
IS	0.94	1.33	1.05
CG	1.47	1.78	1.58
MG	3.90	4.19	3.45
LU	95.05	98.67	109.15
SP	128.29	–	142.16
BT	163.20	168.95	185.13

Table 2. NAS on MPI (4 Nodes, Class A, Intel compiler)

	VAPI(s)	GM (s)	Quadrics (s)
IS	0.83	1.29	1.02
CG	1.46	1.64	1.54
MG	3.20	3.29	2.98
LU	77.51	81.53	77.19
FT	6.20	7.52	7.16
SP	106.69	112.05	103.32
BT	146.31	152.93	146.58

7.4.2 ASCI SWEEP3D Benchmark Results

Table 5 shows the running time of the ASCI SWEEP3D Benchmark on the three MPI implementations. For these tests Intel compilers were used. Two sets of input parameters were for the benchmark. From the table we can see that all three MPI performs comparably. MPI over GM performs a little bit better than others for both sets of input parameters.

Table 3. SWEEP3D on MPI (Intel compiler)

	VAPI(s)	GM (s)	Quadrics (s)
100x100x50	7.45	7.04	7.32
300x300x150	179.18	176.48	176.25

7.5 Related Work

The Message Passing Interface (MPI) has been researched for many years in both industry and universities. MPI [14] has become a de facto standard for developing portable parallel applications. There are a large number of MPI implementations on top of different platforms. Here we only mentioned a few of them, which are closer to our work.

A couple of MPI implementations have been brought up with respect to the emerging network technologies, such as Myrinet [16], Dolphin SCI [9], Quadrics [20], SP networks [2] and VIA networks [7, 3, 13].

InfiniBand architecture, emerging as the leading high performance computing interconnect, has come out as a stable product only in the last two years. Recently, the National Center for Supercomputing Applications (NCSA) at the University of Illinois at Urbana-Champaign reported their MPI implementation atop of the InfiniBand architecture [15]. However, this implementation is on top of the VMI interface. In this paper, we implemented the ADI layer of MPICH directly on top of the InfiniBand Verbs interface, which is different from the NCSA's implementation. In addition, we carry out in-depth, evaluation of our MPI implementation and its comparison with other comtemporary cluster interconnects.

8 Conclusions and Future Work

In this paper we presented design and implementation issues related to supporting MPI on top of InfiniBand's verbs layer. First we introduced the InfiniBand Architecture and described the specifics of Mellanox InfiniBand hardware. Next, we presented an overview of InfiniBand software architecture. Then we presented the design challenges and implementation details associated with putting MPI on top of the Verbs layer. We provided detailed performance evaluation of our MPI implementation with Mellanox's InfiniHost adapters and InfiniScale switch. We also carried out performance comparison with contemporary cluster interconnects such as Myrinet and Quadrics. Our results show that current InfiniBand hardware is capable of delivering significant performance benefits and in some cases even better than Myrinet and Quadrics. InfiniBand products are currently under going rapid growth and optimization. Thus, it is expected that the performance numbers for near future InfiniBand adapters will be even better. These results demonstrate that IBA can be used to build next generation large-scale clusters and data centers with high performance.

In this paper, we have focused only a small subset issues for designing our MPI implementation. The InfiniBand architecture provides many other novel features (atomic operations, multicast support, reliable datagram, service levels, etc.). The firmware releases of current InfiniBand products (such as InfiniHost) do not yet support these features. As new firmware releases enabling these features are rolled-out, we plan to continuously incorporate these features into our design so that we can achieve a high performance and scalable MPI implementation. Our application-level performance evaluation in this paper has been limited to a small testbed. We plan to carry out these experiments in a large-scale testbed. It will also be interesting to analyze how other programming models such as Distributed Shared Memory (DSM) and Get/Put can be designed on top of InfiniBand while exploiting its novel features.

Acknowledgments

We are very grateful to Helen Chen and Curtis Janssen from Sandia National Laboratories, California, for their support of this research under contract #30505. We are extremely grateful to Kevin Deierling, Chanan Shamir, Jeff Kirk, Diego Crupnicoff and Ezra Silvera of Mellanox Inc. for the many helpful discussions we had regarding our InfiniBand Platform. We would also like to thank Darius Buntinas and Weikuan Yu for helping us get the GM and Quadrics performance numbers.

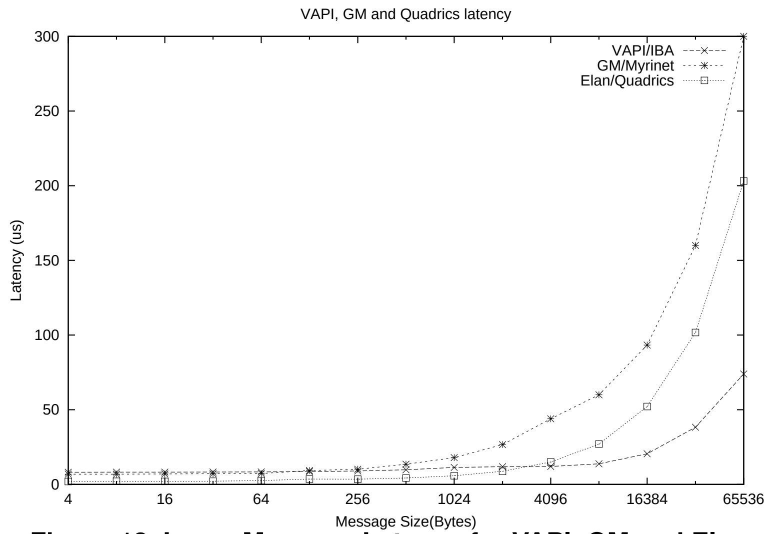

Figure 12. Large Message Latency for VAPI, GM and Elan

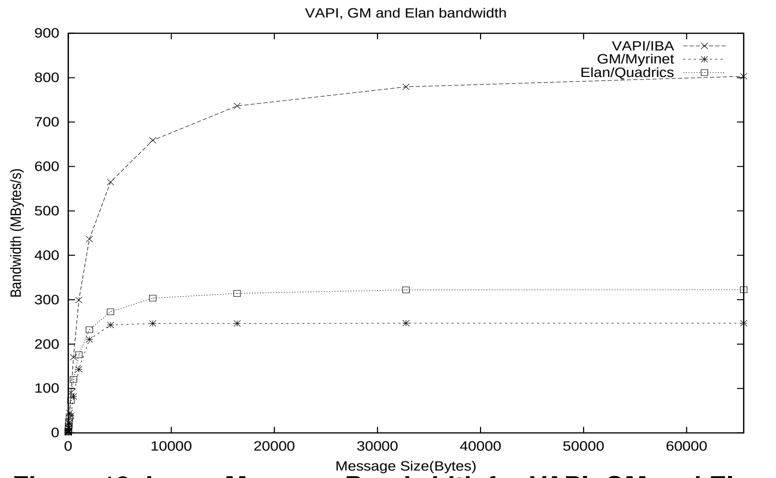

Figure 13. Large Message Bandwidth for VAPI, GM and Elan

References

[1] Mellanox Technologies. http://www.mellanox.com.

[2] M. Banikazemi, R. Govindaraju, R. Blackmore, and D. Panda. MPI-LAPI: An Efficient Implementation of MPI for IBM RS/6000 SP Systems. *IEEE Transactions on Parallel and Distributed Systems*, 12(10):1081–1093, 2001.

[3] M. Bertozzi, M. Panella, and M. Reggiani. Design of a VIA based communication protocol for LAM/MPI suite. In *9th Euromicro Workshop on Parallel and Distributed Processing*, Sept. 2001.

[4] M. Blumrich, C. Dubnicki, E. W. Felten, K. Li, and M. R. Mesarina. Virtual-Memory-Mapped Network Interfaces. In *IEEE Micro*, pages 21–28, Feb. 1995.

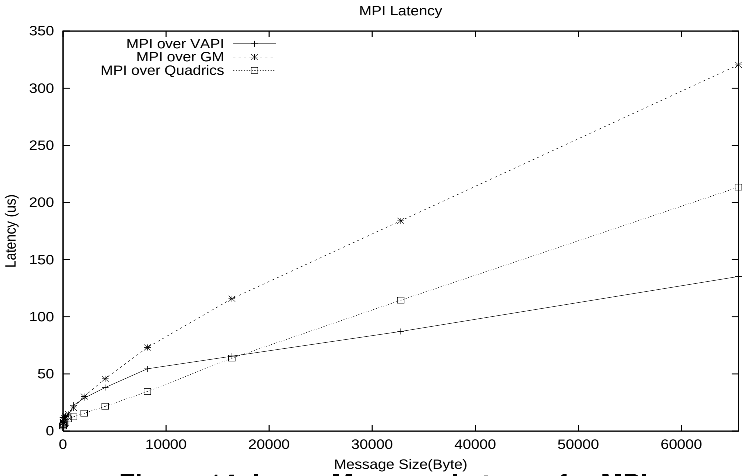

Figure 14. Large Message Latency for MPI

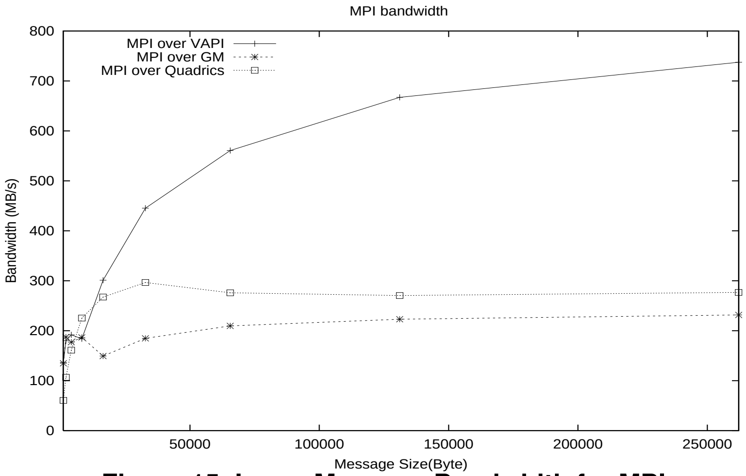

Figure 15. Large Message Bandwidth for MPI

Table 4. NAS on MPI (4 Nodes, Class A, Intel compiler)

	VAPI(s)	GM (s)	Quadrics (s)
IS	0.83	1.29	1.02
CG	1.46	1.64	1.54
MG	3.20	3.29	2.98
LU	77.51	81.53	77.19
FT	6.20	7.52	7.16
SP	106.69	112.05	103.32
BT	146.31	152.93	146.58

Table 5. SWEEP3D on MPI (Intel compiler)

	VAPI(s)	GM (s)	Quadrics (s)
100x100x50	7.45	7.04	7.32
300x300x150	179.18	176.48	176.25

[5] N. J. Boden, D. Cohen, et al. Myrinet: A Gigabit-per-Second Local Area Network. *IEEE Micro*, pages 29–35, Feb 1995.

[6] Compaq, Intel, and Microsoft. VI Architecture Specification V1.0, December 1997.

[7] R. Dimitrov and A. Skjellum. An Efficient MPI Implementation for Virtual Interface (VI) Architecture-Enabled Cluster Computing. http://www.mpi-softtech.com/publications/default.asp, 1998.

[8] D. Dunning, G. Regnier, G. McAlpine, D. Cameron, B. Shubert, F. Berry, A. Merritt, E. Gronke, and C. Dodd. The Virtual Interface Architecture. *IEEE Micro*, pages 66–76, March/April 1998.

[9] Friedrich Seifert and Daniel Balkanski and Wolfgang Rehm. Message Passing Interface Implementation for InfiniBand Architecture. In proceedings of SCI-Europe 2000 held in conjunction with Euro-Par 2000, Munich, Germany., September 2000.

[10] W. Gropp, E. Lusk, N. Doss, and A. Skjellum. A High-Performance, Portable Implementation of the MPI, Message Passing Interface Standard. Technical report, Argonne National Laboratory and Mississippi State University.

[11] H. Tezuka and F. O'Carroll and A. Hori and Y. Ishikawa. Pin-down Cache: A Virtual Memory Management Technique for Zero-copy Communication. In 12th International Parallel Processing Symposium, pages 308–314, Orlando, Florida, Mar 30 - Apr 3, 1998.

[12] InfiniBand Trade Association. InfiniBand Architecture Specification, Release 1.0, October 24 2000.

[13] L. L. N. Laboratory. MVICH: MPI for Virtual Interface Architecture, August 2001.

[14] Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard*, Mar 1994.

[15] Mike Showerman, Bill Magro. Message Passing Interface Implementation for InfiniBand Architecture. http://www.intel.com/idf/us/spr2002/presentations.htm, Spring, 2002.

[16] Myricom. MPICH-GM. http://www.myri.com/myrinet/performance/MPICH-GM/index.html.

[17] NASA. NAS Parallel Benchmarks. http://www.nas.nasa.gov/.

[18] N.J. Boden, D. Cohen, R.E. Felderman, A.E. Kulawik, C.L. Seitz, J.N. Seizovic, and W. Su. Myrinet: A Gigabit-per-second Local Area Network. *IEEE Micro*, 15(1):29–36, February 1995.

[19] S. Pakin, M. Lauria, and A. Chien. High Performance Messaging on Workstations: Illinois Fast Messages (FM). In *Proceedings of the Supercomputing*, 1995.

[20] F. Petrini, W. Feng, A. Hoisie, S. Coll, and E. Frachtenberg. The Quadrics Network: High-Performance Clustering Technology. *IEEE Micro*, 22(1):46–57, 2002.

[21] L. Prylli and B. Tourancheau. BIP: A New Protocol Designed for High Performance Networking on Myrinet. *In Proceedings of the International Parallel Processing Symposium Workshop on Personal Computer Based Networks of Workstations*, 1998. http://lhpca.univ-lyon1.fr/.

[22] G. Shah, J. Nieplocha, J. Mirza, C. Kim, R. Harrison, R. K. Govindaraju, K. Gildea, P. DiNicola, and C. Bender. Performance and experience with lapi - a new high performance communication library for the ibm rs/6000 sp. *International Parallel Processing Symposium*, March 1998.

[23] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra. *MPI–The Complete Reference. Volume 1 - The MPI-1 Core, 2nd edition.* The MIT Press, 1998.

[24] T. von Eicken, A. Basu, V. Buch, and W. Vogels. U-Net: A User-level Network Interface for Parallel and Distributed Computing. In *ACM Symposium on Operating Systems Principles*, 1995.

[25] T. von Eicken, D. E. Culler, S. C. Goldstein, and K. E. Schauser. Active Messages: A Mechanism for Integrated Communication and Computation. In *International Symposium on Computer Architecture*, pages 256–266, 1992.

[26] M. Welsh, A. Basu, and T. von Eicken. Incorporating Memory Management into User-Level Network Interfaces. In *Proceedings of Hot Interconnects V*, Aug. 1997.