High Performance RDMA Based All-to-all Broadcast for InfiniBand Clusters [⋆]

S. Sur, U. K. R. Bondhugula, A. Mamidala, H.-W. Jin, and D. K. Panda

Department of Computer Science and Engineering
The Ohio State University
Columbus, Ohio 43210
{surs, bondhugu, mamidala, jinhy, panda}@cse.ohio-state.edu

Abstract. The All-to-all broadcast collective operation is essential for many parallel scientific applications. This collective operation is called `MPI_Allgather` in the context of MPI. Contemporary MPI software stacks implement this collective on top of MPI point-to-point calls leading to several performance overheads. In this paper, we propose a design of All-to-All broadcast using the Remote Direct Memory Access (RDMA) feature offered by InfiniBand, an emerging high performance interconnect. Our RDMA based design eliminates the overheads associated with existing designs. Our results indicate that latency of the All-to-all Broadcast operation can be reduced by 30% for 32 processes and a message size of 32 KB. In addition, our design can improve the latency by a factor of 4.75 under no buffer reuse conditions for the same process count and message size. Further, our design can improve performance of a parallel matrix multiplication algorithm by 37% on eight processes, while multiplying a 256x256 matrix.

1 Introduction

The Message Passing Interface (MPI) [1] has become the *de-facto* standard in writing parallel scientific applications which run on High Performance Clusters. MPI provides *point-to-point* and *collective* communication semantics. Many scientific applications use collective communication to synchronize or exchange data [2]. The All-to-all broadcast (`MPI_Allgather`) is an important collective operation used in many applications such as matrix multiplication, lower and upper triangle factorization, solving differential equations, and basic linear algebra operations.

InfiniBand [6] is emerging as a high performance interconnect for interprocess communication and I/O. It provides powerful features such as Remote DMA (RDMA) which enables a process to directly access memory on a remote node. To exploit the benefits of this feature, we design collective operations directly on top of RDMA. In this paper we describe our design of the All-to-all Broadcast operation over RDMA which allows us to eliminate messaging overheads like extra message copies, protocol handshake and extra buffer registrations. Our designs utilize the basic choice of algorithms [17] and extend that for a high performance design over InfiniBand.

We have implemented and incorporated our designs into MVAPICH [14], a popular implementation of MPI over InfiniBand used by more than 250 organizations world

[⋆] This research is supported in part by Department of Energy's grant #DE-FC02-01ER25506, National Science Foundation's grants #CNS-0403342, #CNS-0509452 and #CCR-0311542; grants from Intel and Mellanox; and equipment donations from Intel, Mellanox, AMD, Apple and Sun Microsystems.

wide. MVAPICH is an implementation of the Abstract Device Interface (ADI) for MPICH [4]. MVAPICH is based on MVICH [10]. Our performance evaluation reveals that our designs improve the latency of `MPI_Allgather` on 32 processes by 30% for 32 KB message size. Additionally, our RDMA design can improve the performance of `MPI_Allgather` by a factor of 4.75 on 32 processes for 32 KB message size, under no buffer reuse conditions. Further, our design can improve the performance of a parallel matrix multiplication algorithm by 37% on eight processes, while multiplying a 256x256 matrix.

The rest of this paper is organized as follows: in Section 2, we provide a background on the topic. Our motivation is described in Section 3. In Section 4, we describe our RDMA based design in detail. Our experimental evaluation is described in Section 5. Various related works are mentioned in Section 6. Finally, this paper concludes in Section 7.

2 Background

2.1 Overview of InfiniBand Architecture

The InfiniBand Architecture [6] defines a switched network fabric for interconnecting processing and I/O nodes. In an InfiniBand network, hosts are connected to the fabric by Host Channel Adapters (HCAs). InfiniBand utilities and features are exposed to applications running on these hosts through a *Verbs* layer. InfiniBand Architecture supports both channel semantics and memory semantics. In channel semantics, send/receive operations are used for communication. In memory semantics, InfiniBand provides Remote Direct Memory Access (RDMA) operations, including RDMA Write and RDMA Read. RDMA operations are one-sided and do not incur software overhead at the remote side. Regardless of channel or memory semantics, InfiniBand requires that all communication buffers to be "registered". This buffer registration is done in two stages. In the first stage, the buffer pages are pinned in memory (i.e. marked unswappable). In the second stage, the HCA memory access tables are updated with the physical addresses of the pages of the communication buffer.

2.2 `MPI_Allgather` Overview

`MPI_Allgather` is an All-to-all broadcast collective operation defined by the MPI standard [12]. It is used to gather contiguous data from every process in a communicator and distribute the data from the jth process to the jth receive buffer of each process. `MPI_Allgather` is a blocking operation (i.e. control does not return to the application until the receive buffers are ready with data from all processes).

2.3 Related Algorithms and their Cost Models

Several algorithms can be used to implement `MPI_Allgather`. Depending on system parameters and message size, some algorithms may outperform the others. Currently, MPICH [4] 1.2.6 uses the Recursive Doubling algorithm for power-of-two process numbers and up to medium message sizes. For non-power of two processes, it uses the Bruck's algorithm [3] for small messages. Finally, the Ring algorithm is used for large messages [17]. In this section, we provide a brief overview of the Recursive Doubling and Ring algorithms. We will use these algorithms in our RDMA based design.

Recursive Doubling: In this algorithm, pairs of processes exchange their buffer contents. But in every iteration, the contents collected during all previous iterations are also included in the exchange. Thus, the collected information *recursively doubles*. Naturally, the number of steps needed for this algorithm to complete is $log(p)$, where p is the number of processes. The communication pattern is very dense, and involves one half of the processes exchanging messages with the other half. On a cluster which does not have constant bisection bandwidth, this pattern will cause contention. The total communication time of this algorithm is:

$$T_{rd} = t_s * log(p) + (p - 1) * m * t_w \tag{1}$$

Where, t_s = Message transmission startup time, t_w = Time to transfer one byte, m = Message size in bytes and p = Number of processes.

Ring Algorithm: In this algorithm, the processes exchange messages in a ring-like manner. At each step, a process passes on a message to its neighbor in the ring. The number of steps needed to complete the operation is $(p - 1)$ where p is the number of processes. At each step, the size of the message sent to the neighbor is same as the `MPI_Allgather` message size, m. The total communication time of this algorithm is:

$$T_{ring} = (p - 1) * (t_s + m * t_w) \tag{2}$$

3 Can RDMA benefit Collective Operations?

Using RDMA, a process can directly access the memory locations of some other process, with no active participation of the remote process. While it is intuitive that this approach can speed up point-to-point communication, it is not clear how *collective* communications can benefit from it. In this section, we present the answer to this question and present the motivation of using RDMA for collective operations.

3.1 Bypass intermediate software layers

Most MPI implementations [4] implement MPI collective operations on top of MPI point-to-point operations. The MPI point-to-point implementation in turn is based on another layer called the ADI (Abstract Device Interface). This layer provides abstraction and can be ported to several different interconnects. The communication calls pass through several software layers before the actual communication takes place adding unnecessary overhead. On the other hand, if collectives are directly implemented on top of the InfiniBand RDMA interface, all these intermediate software layers can be bypassed.

3.2 Reduce number of copies

High-performance MPI implementations, MVAPICH [14], MPICH-GM [13] and MPICH-QsNet [15] often implement an eager protocol for transferring short and medium-sized messages. In this eager protocol, the message to be sent is copied into internal MPI buffers and is directly sent to an internal MPI buffer of the receiver. This causes two copies for each message transfer. For a collective operation, there are either $2 * log(p)$ or $2*(p-1)$ sends and receives (every send has a matching receive). It is clear that as the number of processes in a collective grows, there are increasingly more and more message copies. Instead, with RDMA based design, messages can be directly transferred without undergoing several copies, as described in section 4.

3.3 Reduce Rendezvous handshaking overhead

For transferring large messages, high-performance MPI implementations often implement the Rendezvous Protocol. In this protocol, the sender sends a `RNDZ_START` message. Upon its receipt, the receiver replies with `RNDZ_REPLY` containing the memory address of the destination buffer. Finally, the sending process sends the `DATA` message directly to the destination memory buffer and issues a `FIN` completion message. By using this protocol, zero-copy message transfer can be achieved.

This protocol imposes bottlenecks for MPI collectives based on point-to-point design. The processes participating in the collective need to continuously exchange addresses. However, these address exchanges are redundant. Once the base address of the collective communication buffer is known, the source process can compute the destination memory address for each iteration. This computation can be done locally by the sending process by calculating the array index for the particular algorithm and iteration number. Thus, for each iteration, RDMA can be directly used without any need for address exchange [16].

3.4 Reduce Cost of Multiple Registrations

InfiniBand [5], like most other RDMA capable interconnects, requires that all communication buffers be registered with the InfiniBand HCA. This "registration" actually involves locking of pages into physical memory and updating HCA memory access tables. After registration, the application receives a "memory handle" with keys which can be used by a remote process to directly access the memory. Thus, for performing each send or receive, the memory area needs to be registered.

Collective operations implemented on top of point-to-point calls would need to issue several MPI sends or receives to different processes (with different array offsets). This will cause multiple registration calls. For current generation InfiniBand software/hardware stacks, each registration has high setup overhead of around 90 μs (section 5). Thus, point-to-point implementation of collectives requires multiple registration calls with significant overhead. However, the RDMA based design would need only *one* registration call. The entire buffer passed to the collective call can be registered in one go. Thus, this will eliminate unnecessary registration calls.

4 Proposed RDMA Based All-to-all Broadcast Design

In this section, we describe our RDMA based design in detail. However, before we use RDMA, we have to deal with several design choices that are posed by the RDMA semantics.

4.1 RDMA Design Choices: Copy Based or Zero Copy

As stated in section 2.1, InfiniBand (like other modern RDMA capable interconnects) requires communication buffers to be registered. Thus, for transferring a message, either (1) the message is copied to a pre-registered buffer, or (2) the message buffer itself is registered at both sender and receiver ends. Approach (2) allows us to achieve zero-copy. Since copy cost is small for smaller messages, approach (1) is used for small messages. As the copy cost is prohibitive for larger messages, approach (2) is used for messages exceeding a certain threshold.

4.2 RDMA-based Design for Recursive Doubling

We propose a RDMA based design for Recursive Doubling (RD) algorithm. In RD, the size of the message exchanged by pairs of nodes doubles each iteration along with the distance between the nodes. If m is the message size contributed by each process, the amount of data exchanged between two processes increases from m in the first iteration to $\frac{mp}{2}$ in the $\log(p)^{th}$ iteration. As we observed in section 4.1, the optimal method to transfer short messages is copy based and for longer messages, we need to use zero copy. However, since in the RD algorithm, the actual message size in each iteration changes, we also have to dynamically switch between copy based and zero copy protocols to achieve an optimal design.

Hence, we switch between the two design alternatives at an iteration k ($1 \leq k \leq \log(p)$) such that the message size being exchanged, $2^{k-1}m$, crosses a fixed threshold M_T. The threshold M_T is determined empirically. Hence, message exchanges in the first k dimensions use a copy-based approach, and those in higher dimensions from $k+1$ through $\log(p)$ use a zero copy approach.

For performing the copy based approach, we need to maintain a pre-registered buffer. We call it “Collective Buffer”. The design issues relating to maintaining this buffer and buffering schemes are described as follows:

Collective Buffer: This buffer is registered at communicator initialization time. Processes exchange addresses of their collective buffers also during that time. Some pre-defined space in the collective buffer is reserved to store the peer addresses and completion flags required for zero-copy data transfers. Data sent in any iteration comprises data received in all previous iterations along with the process’ own message.

Buffering Scheme: In RD, data is always sent from and received to contiguous locations in either the collective buffer or the user’s receive buffer. Since the amount of data written to a collective buffer cannot exceed M_T, the collective buffer never needs to be more than $2M_T$ which is 8 KB (ignoring space for peer addresses and completion flags) for a single Allgather call.

4.3 RDMA Ring for large messages

We implement the Ring algorithm for `MPI_Allgather` over RDMA only for large messages and large clusters. As observed in [17], large clusters may have better near-neighbor bandwidth. Under such scenarios, it is beneficial for `MPI_Allgather` to mainly communicate between neighbors. The Ring algorithm is ideal for such cases. Since we implement this algorithm for only large messages, we use a complete zero copy approach here. The design in this case is much simpler. The benefit of the RDMA-based scheme comes from the fact that we have a single buffer registration and a single address exchange performed by each node instead of p registrations, and $(p-1)$ address exchanges in the point-to-point based design. We use this Ring algorithm for messages larger than 1 MB and process numbers greater than 32.

5 Experimental Evaluation

In this section, we evaluate the performance of our designs. We use three cluster configurations for our tests:

1. *Cluster A:* 32 Dual Intel Xeon 2.66 GHz nodes with 512 KB L2 cache and 2 GB of main memory. The nodes are connected to Mellanox MT23108 HCA using PCI-X 133 MHz I/O bus. The nodes are connected to Mellanox 144-port switch (MTS 14400).
2. *Cluster B:* 16 Dual Intel Xeon 3.6 GHz nodes (EM64T) with 1MB L2 cache and 4 GB of main memory. The nodes are connected to Mellanox MHES18-XT HCA using PCI-Express (x8) I/O bus.
3. *Cluster C:* 8 Dual Intel Xeon 3.0 GHz nodes with 512 KB L2 cache and 2 GB of main memory. The nodes are connected to the same InfiniBand network as Cluster A.

We have integrated our RDMA based design in the MVAPICH [14] stack. We refer to the new design as "MVAPICH-RDMA". The current implementation of `MPI_Allgather` over point-to-point is referred to as "MVAPICH-P2P". Our experiments are classified into three types. First, we demonstrate the latency of our new RDMA design. Secondly, we investigate performance of the new design under low buffer re-use conditions. Finally, we evaluate the impact of our design on a Matrix Multiplication application kernel which uses All-to-all broadcast.

5.1 Latency benchmark for `MPI_Allgather`

In this experiment, we measure the basic latency of our `MPI_Allgather` implementation. All the processes are synchronized with a barrier and then `MPI_Allgather` is repeated 1000 times, using the same communication buffer. The results are shown in Figures 1 and 2 for Cluster A and in Figures 3 for Cluster B. The results from both Clusters A and B follow the same trends. The results are explained as follows:

Small Messages: As described in section 3, the RDMA based design can avoid the various copy and layering overheads in different layers of the MPI point-to-point implementation. The results indicate that latency can be reduced by 17%, 13% and 15% for 16 processes on Cluster A (Fig 1(a)), 32 processes on Cluster A (Fig 2(a)) and 16 processes on Cluster B (Fig 3(a)) for 4 byte message size, respectively.

Medium Messages: For medium sized messages, the point-to-point based design required rendezvous address exchange for transferring messages at every step of the algorithm. However, for the RDMA based `MPI_Allgather`, no such exchange is required (section 4). We note from section 2.3, the number of steps increases as the number of processes, and so does the cumulative cost of address exchange. Our RDMA based design is able to successfully avoid this increasing cost.

The results indicate that latency can be reduced by 23%, 30% and 37% for 16 processes on Cluster A (Fig 1(b)), 32 processes on Cluster A (Fig 2(b)) and 16 processes on Cluster B (Fig: 3(b)) for 32 KB message size, respectively.

Large Messages: Large messages are also transferred using the same zero copy technique used for medium sized messages. Hence, the same address exchange cost can be saved (as described in the previous case). However, since the message sizes are large, the address exchange forms a lesser portion of the overall cost of `MPI_Allgather`. The results for large messages indicate that latency can be reduced by 7%, 6% and 21% for 16 processes on Cluster A (Fig 1(c)), 32 processes on Cluster A (Fig: 2(c)) and 16 processes on Cluster B (Fig 3(c)) for 256 KB message size, respectively.

Scalability: We plot the MPI_Allgather latency numbers with varying process counts, for a fixed message size to see the impact of RDMA design on scalability. Figure 4(a) shows the results for 32 KB message size. We observe that as the number of processes increase, the gap between the point-to-point implementation and RDMA design increases. This is due to the fact that the RDMA design eliminates the need for address exchange (which increases as the number of processes).

5.2 MPI_Allgather latency with no buffer reuse

In the above experiment, we measured the latency of MPI_Allgather when utilizing the same communication buffers for a large number of iterations. The cost of registration was thus amortized over all the iterations by the registration cache maintained by MVAPICH. However, it is not necessary that all MPI applications will always reuse their buffers. In the case where applications use MPI_Allgather with different buffers, the point-to-point based design will be forced to register the buffers separately, thus incurring high cost. The cost of just memory registration is shown in Figure 4(b). We observe that memory registration is in fact quite costly.

In the following experiment, we conduct the same latency test (as mentioned in previous section), but the buffers used for each iteration are different. Figures 5(a) and 5(b) show the results for Clusters A and B, respectively.

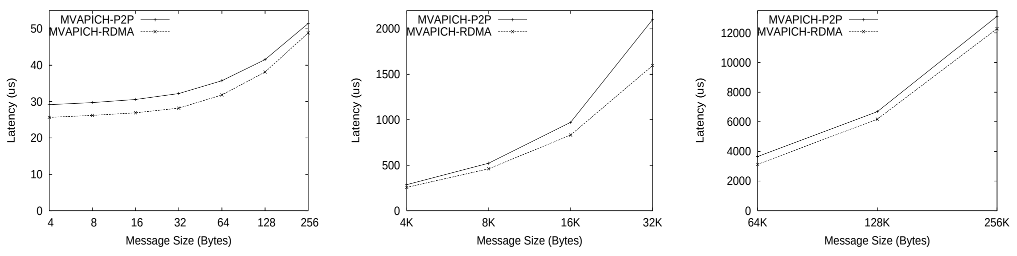
The RDMA based MPI_Allgather performs 4.75 and 3 times better for Cluster A and B for 32 KB message size, respectively.

5.3 Matrix Multiplication Application Kernel

In the previous sections, we have seen how the RDMA design impacts the basic latency of MPI_Allgather. In order to evaluate the impact of this performance boost on end-MPI applications, we build a distributed-memory Matrix Multiplication routine over the optimized BLAS provided by the Intel Math Kernel Library [7]. We use a simple row-block decomposition for the data. In each iteration, the matrix multiplication is repeated with a fresh set of buffers. This application kernel is run on Cluster C using 8 processes. We observe that using our RDMA design, the application kernel is able to perform 37% better for an array size of 256x256, as shown in Figure 5(c).

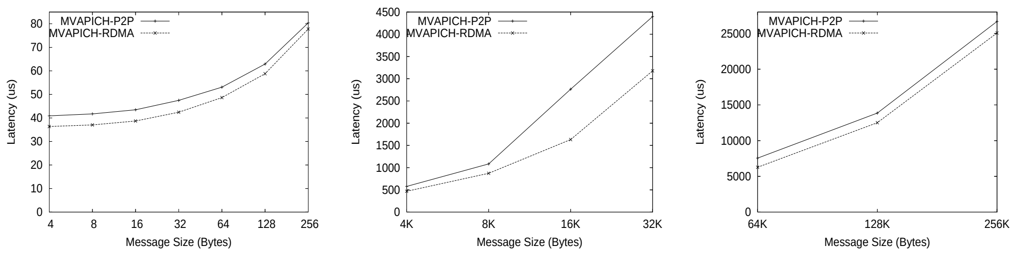
6 Related Work

Recently, a lot of work has been done to improve the performance of collective operations in MPI. In [17] the authors have implemented several well known collective algorithms over the MPI point-to-point primitives. In [9], [16] and [11] the authors have shown the benefits of using RDMA for MPI_Barrier, MPI_Alltoall and MPI_Allreduce collective primitives respectively. In addition, researchers have been focusing on framework for non-blocking collective communication [8]. However our work is different from the above since our work mainly focuses on the All-to-all broadcast of messages using RDMA feature and intelligently choosing the thresholds for copy and zero-copy approaches. Our solution is also according to the MPI specification which require blocking collectives.



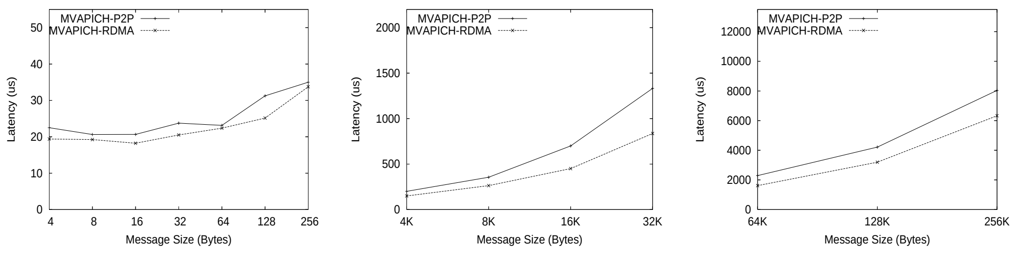
(a) Small Messages (b) Medium Messages (c) Large Messages

Fig. 1. MPI_Allgather Performance on 16 Processes (Cluster A)



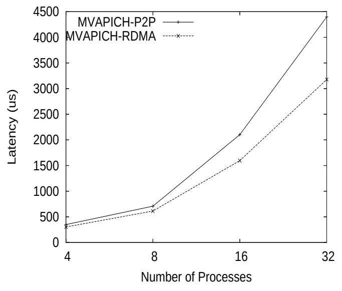
(a) Small Messages (b) Medium Messages (c) Large Messages

Fig. 2. MPI_Allgather Performance on 32 Processes (Cluster A)

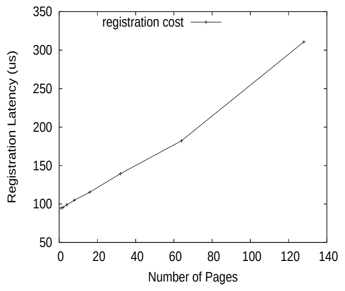


(a) Small Messages (b) Medium Messages (c) Large Messages

Fig. 3. MPI_Allgather Performance on 16 Processes (Cluster B)

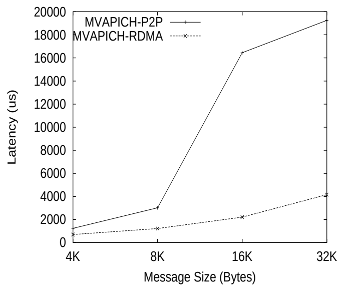


(a) Scalability of RDMA Design for 32 KB message size

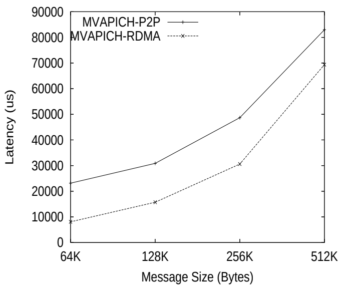


(b) Cost of Registration

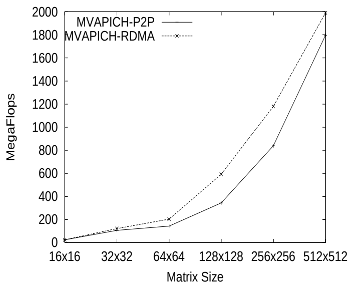
Fig. 4. Scalability and Registration Cost on Cluster A



(a) No Buffer Reuse (Cluster A)



(b) No Buffer Reuse (Cluster B)



(c) Matrix Multiplication

Fig. 5. Impact of Buffer Registration and Performance of Matrix Multiplication

7 Conclusion and Future Work

In this paper, we proposed a RDMA based design for the All-to-all Broadcast collective operation. Our design reduces software overhead, copy costs, protocol handshake – all required by the implementation of collectives over MPI point-to-point. Performance evaluation of our designs reveals that the latency of `MPI_Allgather` can be reduced by 30% for 32 processes and a message size of 32 KB. Additionally, the latency can be improved by a factor of 4.75 under no buffer reuse conditions for the same process count and message size. Further, our design can speed up a parallel matrix multiplication algorithm by 37% on 8 processes, while multiplying a 256x256 matrix.

In the future. We will investigate impact on real world applications in much larger clusters. We will also consider utilizing the RDMA based All-to-all broadcast for designing other collectives like RDMA based All-to-all personalized exchange [16].

References

1. MPI: A Message-Passing Interface Standard. http://www.mpi-forum.org/docs/mpi-11-html/mpi-report.html.
2. D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, D. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber, H. D. Simon, V. Venkatakrishnan, and S. K. Weeratunga. The NAS parallel benchmarks. volume 5, pages 63–73, Fall 1991.
3. J. Bruck, C.-T. Ho, S. Kipnis, E. Upfal, and D. Weathersby. Efficient Algorithms for All-to-All Communications in Multiport Message-Passing Systems. *IEEE Transactions in Parallel and Distributed Systems*, 8(11):1143–1156, November 1997.
4. 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, 1996.
5. InfiniBand Trade Association. InfiniBand Architecture Specification, Volume 1, Release 1.2. http://www.infinibandta.com.
6. InfiniBand Trade Association. InfiniBand Trade Association. http://www.infinibandta.com.
7. Intel Corporation. The Intel Math Kernel Library. http://www.intel.com/cd/software/products/asmo-na/eng/perflib/mkl/index.htm.
8. L. V. Kale, S. Kumar, and K. Vardarajan. A Framework for Collective Personalized Communication. In *International Parallel and Distributed Processing Symposium*, 2003.
9. S. P. Kini, J. Liu, J. Wu, P. Wyckoff, and D. K. Panda. Fast and Scalable Barrier using RDMA and Multicast Mechanisms for InfiniBand-Based Clusters. In *Euro PVM/MPI*, 2003.
10. Lawrence Berkeley National Laboratory. MVICH: MPI for Virtual Interface Architecture. http://www.nersc.gov/research/FTG/mvich/index.html, August 2001.
11. A. Mamidala, J. Liu, and D. K. Panda. Efficient Barrier and Allreduce on IBA clusters using hardware multicast and adaptive algorithms. In *IEEE Cluster Computing*, 2004.
12. Message Passing Interface Forum. *MPI-2: Extensions to the Message-Passing Interface*, Jul 1997.
13. Myricom Inc. Portable MPI Model Implementation over GM, March 2004.
14. Network-Based Computing Laboratory. MPI over InfiniBand Project. http://nowlab.cse.ohio-state.edu/projects/mpi-iba/.
15. Quadrics. MPICH-QsNet. http://www.quadrics.com.
16. S. Sur, H.-W. Jin, and D. K. Panda. Efficient and Scalable All-to-All Exchange for InfiniBand-based Clusters. In *International Conference on Parallel Processing (ICPP)*, 2004.
17. Rajeev Thakur and William Gropp. Improving the Performance of Collective Operations in MPICH. In *Euro PVM/MPI 2003*, 2003.