

# Fast and Scalable Barrier using RDMA and Multicast Mechanisms for InfiniBand-Based Clusters

Sushmitha P. Kini, Jiuxing Liu, Jiesheng Wu, Pete Wyckoff<sup>†</sup>, and Dhabaleswar K. Panda

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

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

Technical Report  
OSU-CISRC-5/03-TR24

# Fast and Scalable Barrier using RDMA and Multicast Mechanisms for InfiniBand-Based Clusters \*

S. P. Kini

J. Liu

J. Wu

P. Wyckoff<sup>†</sup>

D. K. Panda

Computer and Info. Science  
The Ohio State University  
Columbus, OH 43210

{kinis,liuj,wuj,panda}@cis.ohio-state.edu

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

## Abstract

*This paper describes a methodology for efficiently implementing the collective operations, in this case the barrier, on clusters with the emerging InfiniBand Architecture (IBA). IBA provides hardware level support for the Remote Direct Memory Access (RDMA) message passing model as well as the multicast operation. Exploiting these features of InfiniBand to efficiently implement the barrier operation is a challenge in itself. This paper describes the design, implementation and evaluation of three barrier algorithms that leverage these mechanisms. Performance evaluation studies indicate that considerable benefits can be achieved using these mechanisms compared to the traditional implementation based on the point-to-point message passing model. Our experimental results show a performance benefit of up to 1.29 times for a 16-node barrier and up to 1.71 times for non-powers-of-2 group size barriers. Each proposed algorithm performs the best for certain ranges of group sizes and the optimal algorithm can be chosen based on this range. To the best of our knowledge, this is the first attempt to characterize the multicast performance in IBA and to demonstrate the benefits achieved by combining it with RDMA operations for efficient implementations of barrier.*

## 1 Introduction

The barrier operation [15] is a commonly used collective operation in parallel applications that are developed using the Message Passing Interface (MPI) [11] programming model. Barriers are used for synchronizing the parallel processes and involve no transfer of data. They may be used to separate phases of an application program. The `MPI_Barrier` function call is invoked by all the processes in a group. This call blocks a process until all the other members in the group have invoked it. An efficient implementation of the barrier is essential because it is a blocking call and no computation can be performed in parallel with this call. Faster barriers improve the parallel speedup of applications and helps in scalability. Therefore it is important to minimize the time spent waiting on barriers.

The fast improving performance of the modern day interconnects has led to the shift in communication bottleneck from the network fabric to the software layer at the sending and receiving ends. Hence it is vital that the software developers make the best use of the primitives offered by the interconnects and implement the messaging layers with minimal overheads. Communication protocols such as AM [20], LAPI [3], EMP [12], VIA [2] have made user-level transfer of data possible without the need for kernel context switches and multiple data copies. These user level protocols provide a send/receive model of communication which calls for posting of descriptors at the sender and receiver ends. Recent technologies like VIA and InfiniBand Architecture [5] also offer a different model based on memory semantics. They allow transfer of data directly between user level buffers on remote nodes without the active participation of either the sender or the receiver. This method of operation is called Remote Direct Memory Access (RDMA). RDMA allows a process to directly access a remote process' user buffer without the remote process making an explicit function call. There needs to be some initial address exchange done between processes on the endnodes. After this initial handshake, the sends are transparent to the receiver and there is no posting of receive descriptors being done in the critical path.

In the earlier generation MPP and SMP systems, collective operations were achieved by using special hardware support. Today parallel systems are increasingly being built out of affordable commodity workstations and interconnects. These current generation clusters use software based collective operations based on point-to-point communication. In current generation clusters the MPI collective operations are implemented using algorithms that use the MPI point-to-point communication calls. When an operation like barrier is executed the nodes make explicit send and receive calls. The receive operation is generally an expensive operation since it involves posting a descriptor for the message. Also, if the message arrives before the receive call is made, it is placed in a temporary buffer. The receive function then has to search the unexpected queue for the message and when found the data is copied into the user buffer. All this processing adds considerable overhead to the basic send-receive latency, thereby making the entire barrier operation slower. This is the kind of overhead that can be effectively eliminated using RDMA operations.

Another attractive feature in the IBA networks is the support for hardware-based multicast. This primitive is

---

\*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.

provided under the Unreliable Datagram (UD) transport mode. IBA allows processes to attach to a multicast group and then the message sent to the group will be delivered to all the processes in the group. This means that a single descriptor needs to be posted in order to perform a collective operation.

Given these powerful and efficient features in IBA we are faced with the interesting question of whether these remote memory data transfer and multicast support in IBA clusters can be made use of for optimizing the performance of collective operations. While most earlier work focused on optimizing collective operation based on the Send/Recv model, Gupta et al [14] proposed an RDMA based algorithm for VIA-based clusters. The performance characteristics of IBA networks and the added features lead us to rethink the way in which barriers are currently implemented.

In this paper, we aim to provide answers to the following two questions:

1. *Can we optimize the MPI collective operations by using algorithms that leverage the RDMA primitives in IBA instead of algorithms that use the existing MPI point-to-point operations?*

2. *Can the multicast primitives in IBA be used to implement scalable collective communication operations?*

The paper shows that replacing the point-to-point communication calls in the collective operations with faster lower-level operations can provide significant performance gains. Performance improvement is possible due to various reasons. Primarily, the number of data copies is reduced by avoiding point-to-point messaging protocols. Also, software overheads like tag matching and unexpected message handling are eliminated. The hardware multicast feature fits in well with the semantics of collective operations and hence can be utilised to our advantage.

We propose three algorithms that utilize these features in IBA. These algorithms were implemented and integrated into the MVAPICH [7] implementation of MPI over IBA, and we discuss the design and implementation issues. We also present the results of our performance evaluations on two different clusters, and show that considerable benefits are achieved using the proposed techniques.

The rest of the paper is organized as follows: Section 2 gives an overview of IBA, and MVAPICH. In Section 3 we describe the proposed algorithms. Section 4 discusses the issues involved in design and in Section 5 we describe the details of the implementation. Section 6 presents the experimental results. We mention some related work in Section 7 and conclude in Section 8.

## 2 Overview of IBA and MVAPICH

In this section we provide an overview of InfiniBand Architecture and the set of features that can be utilized for the efficient implementation of point-to-point and collective message passing operations. We also provide a brief overview of the MVAPICH message passing library.

### 2.1 InfiniBand Architecture

InfiniBand Architecture [5] is emerging as the next generation interconnect for I/O and inter-process communication. This industry standard uses scalable

switched serial links to design clusters and servers offering high bandwidth and low latency. IBA makes use of kernel-bypass techniques to offer zero-processor copy data transfers between user level processes on remote nodes. In an InfiniBand network, nodes are connected to the IBA fabric using Channel Adapters (CA). Host Channel Adapters (HCA) are installed into the processing nodes and initiate communication within the fabric. Target Channel Adapters (TCA) connect I/O nodes to the fabric.

IBA defines a semantic interface called Verbs to configure, manage and operate a HCA. VAPI is the Verbs implementation provided by Mellanox Technologies [9] for the HCAs. It supports two kinds of communication semantics: channel semantics and memory semantics. In channel semantics, send/receive operations are used for communication. In memory semantics, remote direct memory access operations (RDMA write and RDMA read) are used. Currently, two types of transport services, Reliable Connection (RC) and Unreliable Datagram (UD) are implemented by InfiniHost. In order to communicate, each process creates a Queue Pair (QP) which consists of a Send Queue and a Receive Queue. The transport service needed has to be specified when the QPs are created. Communication requests are initiated by posting Work Queue Requests (or descriptors) to the work queues. The HCA executes these work requests in the order that they are placed in the work queue. When the HCA completes a request it places a Completion Queue Entry (CQE) in the Completion Queue (CQ). The processes can then poll on the CQs to check for completion of the requests. User buffers used for transferring data must be registered first before they can be used for communication.

Addressing of the destination endpoints is determined based on the QP service type. For RC QPs, the destination address is specified when the QP is created. For UD QPs, the destination address of the node is contained in the address handle which is submitted in the work request.

Some of the IBA features that are of interest in the message passing context are described below.

#### 2.1.1 RDMA Read

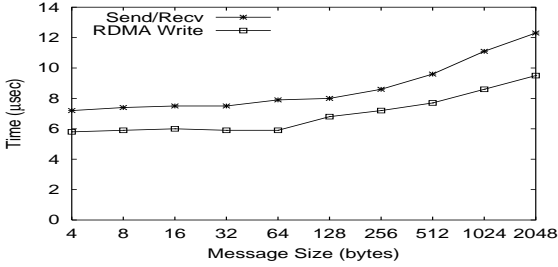
This is a memory semantic operation that allows a process to read a virtually contiguous buffer on a remote node and write to a local memory buffer. RDMA service is available only on the RC transport mode.

#### 2.1.2 RDMA Write

This memory semantic operation allows a process to write to a virtually contiguous buffer on a remote node. This is a one-sided operation that does not incur a software overhead at the remote side. There is also a gather list available to send data from non-contiguous local buffers. Figure 1 shows the difference in latencies between the Send/Receive and RDMA write operations as measured on Cluster 1, described in Section 6. We see that the performance of RDMA write is much better than that of a Send/Receive operation and hence it would be beneficial to use this primitive for our purposes.

#### 2.1.3 Multicast

Multicast is the ability to send a single message to a specific address and have it delivered to multiple processes



**Figure 1. Comparison of RDMA write latency with Send/Recv latency at the VAPI layer (Cluster 1)**

which may be on different endnodes. This feature is implemented as a combination of the capabilities of the IBA HCA, switch and software layers by replicating the multicast message and sending it to all the designated receivers. Performance evaluations of this multicast primitive with the InfiniHost HCAs, InfiniScale switch and VAPI show that it takes about  $9.6\mu\text{s}$  to send a 1-byte message to 1 node and  $9.8\mu\text{s}$  to send the message to 7 nodes. This shows that the operation is very scalable and can be used effectively to design scalable collective operations. The multicast facility is available only with the UD service type. The UD service is connectionless and unacknowledged. It allows the consumer of the QP to communicate with any UD QP on any node, and thus greatly improves the scalability of IBA. Current version of VAPI supports a single multicast group that includes all the nodes in the subnet. We have made use of this “broadcast” primitive for our implementation. When the later versions of VAPI provide support for attaching to different multicast groups it will be possible to design new algorithms where messages are sent only to a select group of processes.

## 2.2 Message Passing Interface

Message Passing Interface (MPI) is a standard library specification for message passing in parallel applications. It defines operations for both point-to-point and collective communication. MPICH [4] from Argonne National Laboratory is a popular open source implementation of the MPI standard. At the core of the MPICH design is a small set of functions that form the Abstract Device Interface (ADI) [16]. The ADI allows easy porting of MPICH to various interconnect technologies. MVICH [6] from Lawrence Berkeley National Laboratory is one such implementation of ADI2 for VIA based clusters. MVAPICH [7] is the implementation of ADI2 for the VAPI interface of the InfiniHost HCAs and is derived from MVICH. Our barrier design, implementation and evaluations in this paper have been done using the MVAPICH 0.8.5 code base.

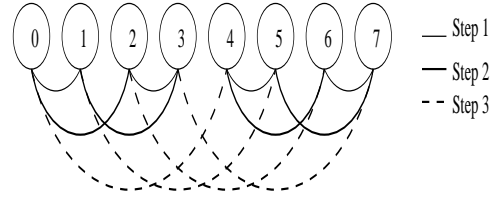
## 3 Proposed Barrier Algorithms

In this section we describe the three algorithms that we have designed and implemented for the barrier operation. The aim is to leverage the fastest protocols (RDMA and multicast) offered by IBA to the fullest extent possible. In the following subsections we denote processes using symbols  $i, j, k$  and the total number of processes involved in the barrier is denoted by  $N$ . We refer to the process that has a distinguished role to play

in some algorithms as the *root*. We indicate the number of the current barrier by the symbol *barrier id*.

### 3.1 RDMA-based Pairwise Exchange (RPE)

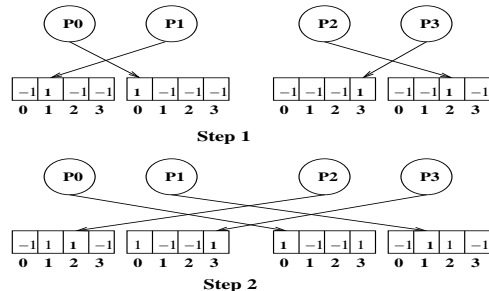
The algorithm for the barrier operation in the MPICH distribution is called the Pairwise Exchange (PE) recursive doubling algorithm. MPICH makes use of the MPI\_Send and MPI\_Recv calls for the implementation of this algorithm. This is a recursive algorithm where the nodes are paired up and each node does a send and receive with its partner. If the number of nodes performing the barrier is a power of two, then every process in this algorithm makes  $\log_2 N$  sends and receives, and thus takes  $\log_2 N$  steps. If  $N$  is not a power of two, then the algorithm takes  $\lfloor \log_2 N \rfloor + 2$  steps. Figure 2 shows the steps performed for a 8 node barrier.



**Figure 2. Pairwise Exchange Algorithm**

Now we describe how this algorithm can be performed using the RDMA Write primitive. The barrier is a collective call, and so each process keeps a running count of the current barrier number, *barrierid*. Each process has an array of bytes of length  $N$ . In each step of the PE, process  $i$  writes the *barrierid* in the  $i^{\text{th}}$  position of the array of the partner process  $j$ . It then waits for the *barrierid* to appear in the  $j^{\text{th}}$  position of its own array. Since each process is directly polling on memory for the reception of data, it avoids the overhead of posting descriptors and copying of data from temporary buffers, as is the case when the MPI\_Recv call is used.

Figure 3 gives a pictorial representation of this algorithm. Here  $N$  is 4, and the processes are called P0, P1, P2, and P3. In the first step P0 does an RDMA write of *barrierid*, in this case 1, to index 0 of P1’s array and waits for P1 to write in index 1 of its own array. In the second step it performs the same operations with P2.



**Figure 3. Steps performed in RPE for a 4-node barrier**

### 3.2 RDMA-based Dissemination (RDS)

In this Dissemination Barrier algorithm as described in [10], the synchronization is not done pairwise as in the

previous algorithm. In round  $m$ , process  $i$  sends a message to process  $j = (i + 2^m) \bmod N$ . It then waits for a message from the process  $k = (i + N - 2^m) \bmod N$ . This algorithm takes  $\lceil \log_2 N \rceil$  steps at each process, irrespective of whether there are power of two or non-power of two number of nodes and thus is a more efficient pattern of synchronizations.

The barrier signaling operations using RDMA write are done exactly as in the RPE algorithm, and this algorithm only varies in way in which the processes are grouped for communication in each step.

The Figure 4 shows the communication in the various steps of this algorithm for a barrier on 5 nodes.

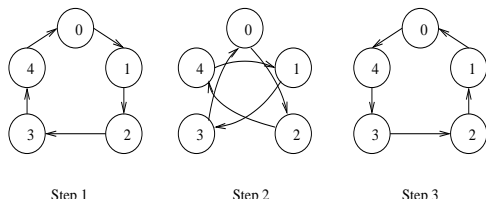


Figure 4. Dissemination Algorithm

### 3.3 RDMA-based Gather and Multicast (RGM)

In this scheme, the barrier operation is divided into two phases. In the first phase called the gather, every node indicates its arrival at the barrier by sending a message to a special process, *root*. This process of gather can be done in a hierarchical fashion by imposing a logical tree structure on the processes. Once *root* has received the messages from all its children, it enters the multicast phase. In this phase *root* broadcasts a message to all the nodes to signal that they can now exit the barrier.

In this two-step technique we use RDMA writes in the gather phase. The processes are arranged in a tree structure. Each process has an array of bytes on which it polls for messages from its children. Once it receives messages from all its children, the process forwards the message to its parent.

When *root* receives all the RDMA messages, it does a hardware multicast to all the processes. The multicast message contains the *barrierid*. This phase is a one step process, since the multicast primitive is such that the single message gets sent to all the members of the multicast group.

Let us assume that the gather phase is done with a maximum fan-in of  $l$ . The value of  $l$  is chosen to be a ( $powerof2-1$ ) value, and  $l < N$ . So the number of levels in the tree created in this phase will be  $\lceil \log_{l+1} N \rceil$ , and this is the number of hops done by the barrier signal to reach *root*. In the multicast phase just one step is taken by the *root* to signal completion of the barrier to all nodes.

Figure 5 shows how this algorithm works for a barrier on 8 processes. Here the gather is done using a 2 stage tree with the value of  $l$  as 3. Process 0 is *root*. The value for  $l$  can be chosen based on the number of nodes and the performance of the RDMA write operation.

## 4 Design Issues

We now discuss the intrinsic issues associated with the design and implementation of the proposed algo-

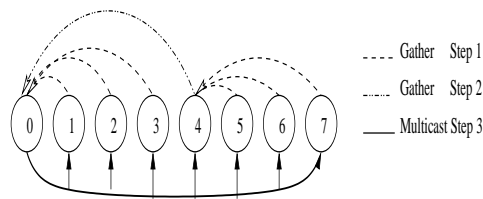


Figure 5. Gather and Multicast Algorithm

rithms. We discuss the buffer management and data reception issues to be handled when using RDMA. We also discuss the techniques to add reliability for the UD multicast. In this section we present different alternatives for each of these issues. In the next section, we focus on our choices and their implementations.

### 4.1 Buffer Management

IBA specification requires that all the data transfer be done only between buffers that are registered. Implementing collective operations on top of point-to-point message passing calls leads us to rely on the internal buffer management and data transfer schemes which might not always be optimal in the collective operations context. In using RDMA, we have better control of the buffer consumption patterns. In order to use the RDMA method of data transfer, each node is required to pin some buffers and send/receive data using them. Also, the remote nodes should be aware of the local buffer address and memory handle, which means that a handshake for the address exchange should be done. The allocation and registration can be done at various stages during the life of the MPI application. The first option is for each process to allocate a set of buffers for the barrier operation and exchange the addresses during the initialization phase(i.e, as a part of MPI\_Init). The disadvantage of this approach is that the buffers will be allocated and pinned even if the application is not making any barrier call.

The second option is for the buffers to be registered during the first barrier call. This means that the buffers are allocated only when needed. However, buffer registration is an expensive process and therefore the first barrier call is bound to take more time.

Another alternative is to allow dynamic allocation and registration of buffers during every barrier call. But this calls for an address exchange operation during each barrier call.

### 4.2 Data Reception

The RDMA write operation is transparent at the receiving end and hence the receiver is not aware of the arrival of data. We need a mechanism to notify the receiver of the completion of the RDMA write.

One method is to make use of the RDMA with immediate data feature. This operation consumes a descriptor at the remote end and hence deprives us of the transparency benefit of the RDMA write.

The other method is for the receiver to poll on the buffers for arrival of data. This means that when the buffers are allocated, they will need to be initialized with some special data so that the data arrival can be recognized.

### 4.3 Adding reliability for unreliable multicast operations

The MPI specification assumes that the underlying communication interface is reliable and that the user need not cope with communication failures. Since the multicast operation in IBA is unreliable, reliability has to be handled in our design.

One alternative is to provide an acknowledgment (ACK) message from the processes after every multicast message is received. The sending process waits for the ACKs from all the nodes and retransmits otherwise. This technique is very expensive since there is a message being sent back from every process to the root, even after the barrier is logically completed.

Another technique would be for each receiving process to maintain a timer and send a negative acknowledgment (NAK) when it has not received a message. This NAK could be sent using the Send/Recv primitives. When the root process receives this message, it can retransmit the multicast message. However, this means that the application should make some MPI communication call in order for the root node to receive the packet and make progress.

The IB specification allows for event handlers to be called when a completion queue entry is generated. There is the option of triggering these event handlers on the receive side only if the “solicit” flag is set in the message by the sender. This facility can be used in the NAK message. By setting the solicit flag, this message triggers the event handler at the *root*, which then does a retransmission of the multicast message.

## 5 Implementation Details

We now describe the design decisions made and the implementation details for the three algorithms.

### 5.1 Buffer Management and Address Exchange

Since the barrier is a collective call, during the first MPIBarrier call, all the processes allocate memory for the barrier. The size of the memory allocated is the same as the size of the communicator. Each element in this allocated array will be written by the corresponding process using an RDMA write call. Since every process in the communicator is identified by a *rank* the array elements can be indexed using this *rank* value.

In order to perform an RDMA write, a process needs to provide the remote memory’s virtual address and the memory handle that is obtained after the registration of the memory. After the allocation of the buffers, the nodes exchange these addresses and memory handles. This address exchange happens using the send and receive primitives.

This design option seems to be the best among the ones mentioned in Section 4, since it ensures that the memory is registered only if the application is involved in collective operations. The overhead is also not increased since the time for address exchange will always be spent, either during the initialization phase, or in the first barrier as is done currently. Once the buffers are allocated, they can be used for all the barriers executed during the life of the process.

### 5.2 Data validation at the receiving end

There is a static count called the *barrierid* that is maintained by each process. This value is always positive. So during the initialization we assign a negative value to all the array elements. When a process needs a message from a remote process, it polls the corresponding array element. It waits for the value to be greater than or equal to the current *barrierid*. This is needed to handle cases with consecutive barriers. If one process is faster than the other, it will enter the second barrier before the other can exit the first one. Thus it will write the larger barrier number in the array. This design alternative seems to give the best performance among all those mentioned in Section 4.

### 5.3 Handling UD multicast messages along with the RC messages

In the broadcast phase of the RGM algorithm we make use of the multicast feature which uses UD packets. This requires that every process create a QP for the UD service type. This is done during MPI\_Init. The address of the QPs is also exchanged among all the processes. The QP of each process is also attached to the global multicast group in order to enable it to receive messages sent to the multicast group.

In the IB specification the UD messages have the initial 40 bytes assigned for the Global Route Header (GRH). If the RC and UD completions are sent to the same CQ, it will be difficult to distinguish between RC and UD messages based on the data content because we don’t know where the actual data starts. Hence we create a separate CQ for the UD work request completions.

### 5.4 Reliability

Once a process sends the barrier message to its parent in the gather phase, it begins to wait for the multicast message from the *root*. We impose a timeout on this phase, and send a NAK to the *root* if no message is received within the time period specified. The NAK message is sent using the Send primitive and it contains the “solicit” flag set to true. The NAK message also contains the *barrierid* that the process is currently waiting to complete.

When the NAK message arrives at the *root*, it triggers the registered completion event handler. The *root* then checks if the message is valid retransmit request by looking at the barrier number. It then does a retransmit of the multicast message for that barrier number.

We have seen in our clusters that the rate of dropping UD packets is very low, and hence this reliability feature is not called upon often. Also, since IBA allows us to specify service levels to QPs, we could assign high priority service levels to the UD QPs. Thus the chances of these messages getting dropped is reduced even further. We also see that in the normal scenarios where there are no packets dropped, there is no overhead imposed by the reliability component.

## 6 Performance Evaluation

We conducted our performance evaluations on the following set of clusters.

Cluster 1 : A cluster of 8 SuperMicro SUPER P4DL6 nodes, each with dual Intel Xeon 2.4GHz processors, 512MB memory, PCI-X 64-bit 133MHz bus, and connected to a Mellanox InfiniHost MT23108 DualPort 4x HCA. The nodes are connected using the Mellanox InfiniScale MT43132 eight 4x port switch. The Linux kernel version is 2.4.7-10smp. The InfiniHost SDK version is 0.1.2 and the HCA firmware version is 1.17.

Cluster 2 : A cluster of 16 Microway nodes, each with dual Intel Xeon 2.4GHz processors, 2GB memory, PCI-X 64-bit 133MHz bus, and connected to a Topspin [17] InfiniBand 4x HCA [18]. The HCAs are connected to the Topspin 360 Switched Computing System, which is a 24 port 4x InfiniBand switch with the ability to include up to 12 gateway cards in the chassis. The Linux kernel version is 2.4.18-10smp. The HCA SDK version is 0.1.2 and firmware version is 1.17.

The barrier latency was obtained by executing `MPIBarrier` 1000 times and the average of the latencies across all the nodes was calculated.

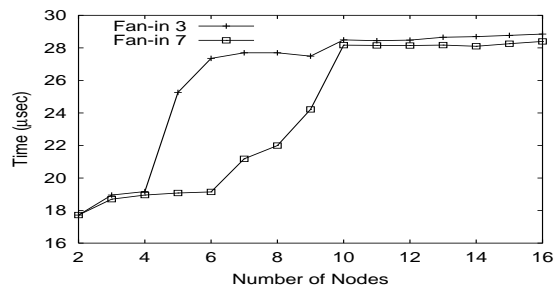
Figure 7 and 8 show the performance comparisons of the three proposed barrier algorithms with MPI-PE, the standard pairwise exchange MPICH implementation of the barrier. On the left-hand side we show the absolute values of the barrier latency and on the right-hand side we show the factors of improvement.

In Figure 7(a) and 7(c), we see that RPE and RDS perform better than MPI-PE. For group sizes of 2 and 4, RGM does worse because the base latency of the UD multicast operation is greater than that of a single RDMA write. The performance of RPE and RDS for powers-of-2 group sizes is very similar. For 8 nodes in Cluster 1, we gain as much as 1.25 factor of improvement with RPE and 1.27 with RDS, as seen in Figure 7(b). Figure 7(d) shows that in Cluster 2, RGM does the best for 16 nodes with an improvement of 1.29. This is because for larger group sizes, RGM has the benefit of the constant time multicast phase.

Figure 8 illustrates the performance gains obtained for all group sizes. We see that the pairwise exchange algorithms, MPI-PE and RPE, always penalize the non-power-of-2 cases, and this is not seen in RDS and RGM. Hence on Cluster 1, RDS and RGM gain a performance improvement of up to 1.64 and 1.71 respectively. On Cluster 2, we see that RGM performs best in most cases and the maximum factor of improvement seen is 1.59. We see that the factor of improvement for RPE is almost a constant in all cases because the benefit is obtained by the constant difference in the latency between a point-to-point send/receive operation and an RDMA-Write/poll operation.

The performance of the RGM algorithm varies with the values for maximum fan-in in the gather phase. As this value decreases, the height of the tree increases and this will increase the number of RDMA writes being done. But if this value is large, the parent node becomes a hot-spot, that could possibly cause degradation in performance. Hence we need to choose the optimal value for the fan-in. From Figure 6, we see that the fan-in value of 7 performs the best. Hence for all our performance evaluations we choose this value in the RGM implementation.

As mentioned earlier, the pairwise exchange algorithm does badly for non-power-of-2 group sizes because of 2 extra operations. Hence in order to do a fair



**Figure 6. Performance of RGM algorithm for varying fan-in values on Cluster 2**

comparison, we implemented the Dissemination algorithm with the point-to-point MPI functions. We refer to this as MPI-DS. The barrier latencies of the proposed algorithms are better than that of MPI-DS too. Figure 9 shows the comparison of the RDS and RGM implementations with MPI-DS. It is to be noted that inspite of providing benefits to the current MPI implementation, RDS achieves up to 1.36 factor of improvement, and RGM achieves 1.46 on Cluster 1. We see an improvement of 1.32 with RDS and 1.48 with RGM on Cluster 2.

## 7 Related Work

The benefits of using RDMA for point-to-point message passing operations for IBA clusters has been described in [8]. The methods and issues involved in implementing point-to-point operations over one-sided communication protocols in LAPI are presented in [1]. However using these optimized point-to-point operations does not eliminate the data copy, buffering and tag matching overheads. A lot of research has taken place in the past to design and develop optimal algorithms for collective operations on various networks using point-to-point primitives, but not much work has been done on selection of the communication primitives themselves.

RDMA based design of collective operations for VIA based clusters [13, 14] has been studied earlier. Combining remote memory and intra-node shared memory for efficient collective operations on IBM SP has been presented in [19]. None of these papers focus on taking advantage of novel mechanisms in IBA to develop efficient collective operations.

## 8 Conclusions and Future Work

In this paper, we have presented three new approaches (RPE, RDS, and RGM) to efficiently implement the barrier operation on IBA-based clusters while taking advantage of the RDMA and multicast functionalities of IBA. The experimental results we achieved on 8 and 16 node clusters show that the proposed approaches significantly outperform the current barrier implementations in MPI that use point-to-point messaging. The RGM scheme tends to perform well for larger group sizes, while RPE and RDS perform better for smaller groups. The results also show that the schemes are scalable with system size and will provide better benefits for larger clusters. Therefore we arrive at the conclusion that the efficiency of the barrier operations can be considerably improved compared to the traditional point-to-point messaging calls based implementations by using the novel mechanisms of IBA.

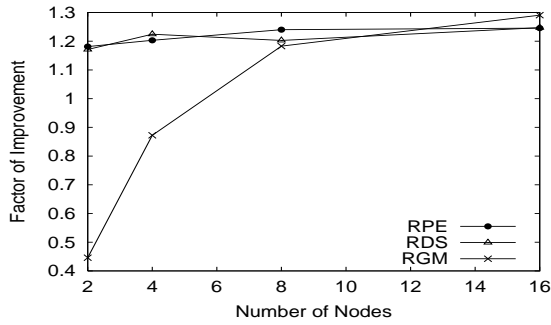
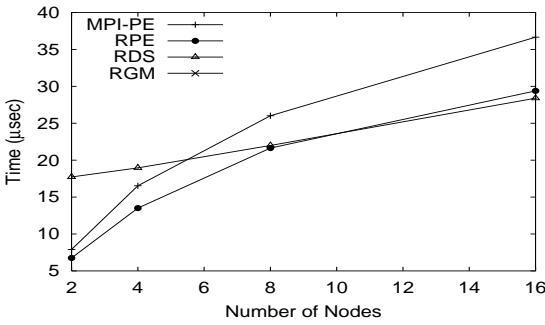
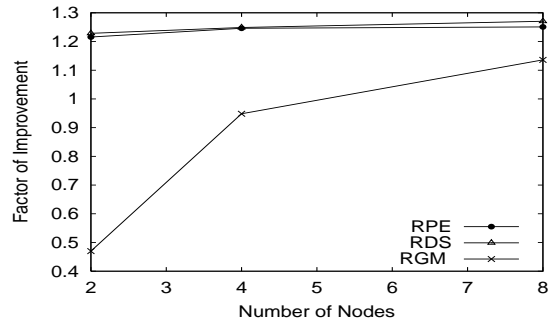
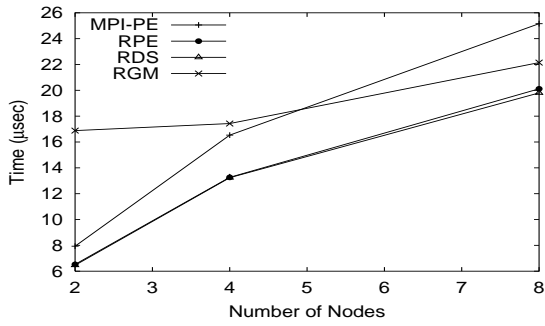


Figure 7. Comparison of MPI-PE with the proposed algorithms for power-of-2 group size on Clusters 1 and 2

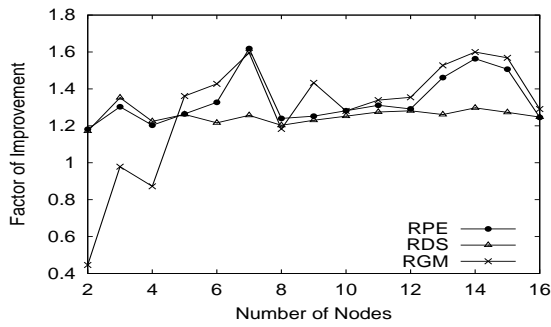
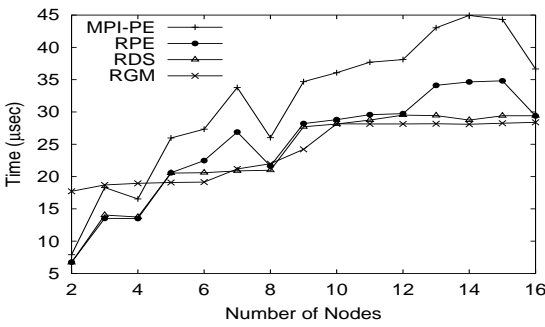
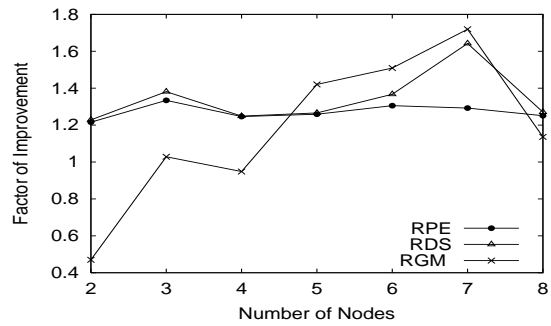
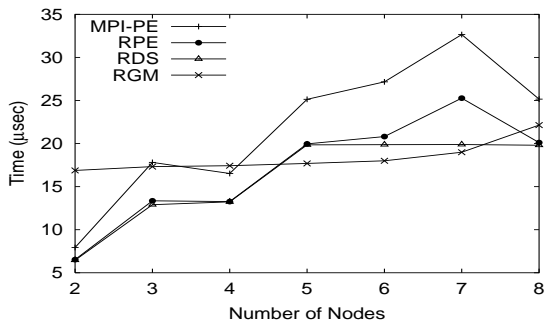


Figure 8. Comparison of MPI-PE with the proposed algorithms for all group sizes on Clusters 1 and 2

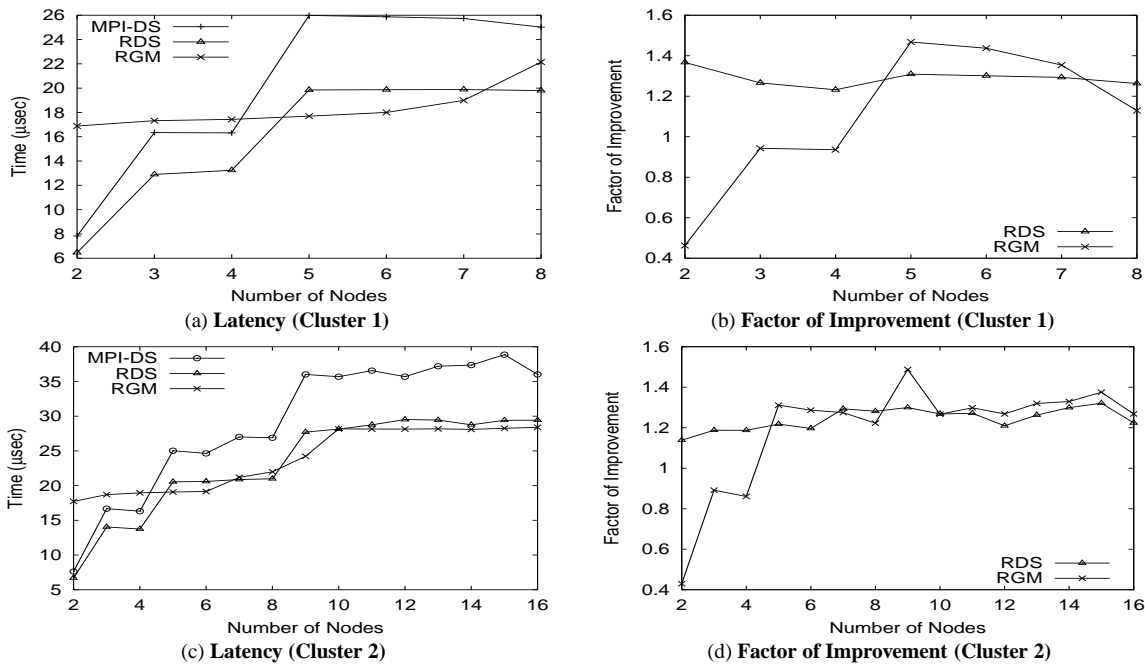


Figure 9. Comparison of MPI-DS with the proposed algorithms on Clusters 1 and 2

This paper concentrated on the barrier operation. We are also working on extending these ideas to implement other collective operations like broadcast and allreduce. We expect the challenges and scope for improvement to be greater in these cases because of the data transfer involved in these operations. We are also planning to use the other features of IBA, like support for atomic and RDMA read operations to implement the collective operations efficiently.

### Acknowledgments

We would like to thank Kevin Deierling, Jeff Kirk, and Ezra Silvera from Mellanox Technologies for their support with the InfiniBand hardware and software. We would also like to thank Ben Eiref, Robert Starmer, and Lorne Boden from Topspin Communications for all their efforts in providing us access to their 16 node InfiniBand cluster. We would like to thank Rinku Gupta, Amith Rajith Mamidala, Pavan Balaji and Balasubramaniam Chandrasekaran for their help and valuable suggestions.

### References

- [1] Mohammad Banikazemi, Rama K. Govindaraju, Robert Blackmore, and Dhabaleswar K. Panda. MPI-LAPI: An Efficient Implementation of MPI for IBM RS/6000 SP Systems. *IEEE TPDS*, pages 1081–1093, October 2001.
- [2] Compaq, Intel, and Microsoft. VI Architecture Specification V1.0, December 1997.
- [3] G. H. 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. *IPPS '98*, March 1998.
- [4] William Gropp, Ewing Lusk, Nathan Doss, and Anthony Skjellum. High-performance, portable implementation of the MPI Message Passing Interface Standard. *Parallel Computing*, 22(6):789–828, 1996.
- [5] InfiniBand Trade Association. InfiniBand Architecture Specification, Release 1.0, October 24 2000.

- [6] Lawrence Berkeley National Laboratory. MVICH: MPI for Virtual Interface Architecture. <http://www.nersc.gov/research/FTG/mvich/index.html>, August 2001.
- [7] Jiuxing Liu, Jiasheng Wu, Sushmitha P. Kini, Darius Buntinas, Weikuan Yu, Balasubraman Chandrasekaran, Ranjit Noronha, Peter Wyckoff, and Dhabaleswar K. Panda. MPI over InfiniBand: Early Experiences. Technical Report, OSU-CISRC-10/02-TR25, January 2003.
- [8] Jiuxing Liu, Jiasheng Wu, Sushmitha P. Kini, Pete Wyckoff, and Dhabaleswar K. Panda. High Performance RDMA-Based MPI Implementation over InfiniBand. In *ICS '03*, June 2003.
- [9] Mellanox Technologies. Mellanox Technologies.
- [10] John M. Mellor-Crummey and Michael L. Scott. Algorithms for scalable synchronization on shared-memory multiprocessors. *ACM ToCS*, 9(1):21–65, 1991.
- [11] Message Passing Interface Forum. MPI: A Message Passing Interface. In *Supercomputing '93*, pages 878–883. IEEE Computer Society Press, 1993.
- [12] P. Shivam, P. Wyckoff and D. K. Panda. EMP:Zero-copy OS-bypass NIC-driven Gigabit Ethernet Message Passing. In *Supercomputing '01*, November 2001.
- [13] R. Gupta, P. Balaji, D. K. Panda, and J. Nieplocha. Efficient Collective Operations using Remote Memory Operations on VIA-Based Clusters. In *IPDPS '03*, April 2003.
- [14] R. Gupta, V. Tipparaju, J. Nieplocha and D. K. Panda. Efficient Barrier using Remote Memory Operations on VIA-Based Clusters. In *Cluster 02*, September 2002.
- [15] 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.
- [16] Rajeev Thakur, William Gropp, and Ewing Lusk. An Abstract-Device Interface for Implementing Portable Parallel-I/O Interfaces. In *Frontiers '96*. IEEE Computer Society, Oct 1996.
- [17] Topspin Communications, Inc. <http://www.topspin.com/>.
- [18] Topspin Communications, Inc. Topspin InfiniBand Host Channel Adapter, <http://www.topspin.com/solutions/hca.html>.
- [19] V. Tipparaju, J. Nieplocha, D. K. Panda. Fast Collective Operations Using Shared and Remote Memory Access Protocols on Clusters. In *IPDPS '03*, April 2003.
- [20] T. von Eicken, D. E. Culler, S. C. Goldstein, and K. E. Schauer. Active Messages: A Mechanism for Integrated Communication and Computation. In *ISCA*, pages 256–266, 1992.