Fast and Scalable Barrier using RDMA and Multicast Mechanisms for InfiniBand-Based Clusters [⋆]

Sushmitha P. Kini[1], Jiuxing Liu[1], Jiesheng Wu[1], Pete Wyckoff[2], and Dhabaleswar K. Panda[1]

[1] Dept. of CIS, The Ohio State University, Columbus, OH - 43210
{kinis, liuj, wuj, panda}@cis.ohio-state.edu
[2] Ohio Supercomputer Center, 1224 Kinnear Road,
Columbus, OH - 43212
pw@osc.edu

Abstract. This paper describes a methodology for efficiently implementing the barrier operation, 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. 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. This framework has significant potential for developing scalable collective communication libraries for IBA-based clusters.

1 Introduction

Barriers are used for synchronizing the parallel processes in applications based on the Message Passing Interface (MPI) [11] programming model. 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.

[⋆] 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.

Recent communication technologies like VIA and InfiniBand Architecture [3] offer a model of data transport 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 is a one-sided operation that does not incur a software overhead at the remote side. This method of operation is called Remote Direct Memory Access (RDMA).

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. This overhead can be effectively eliminated using RDMA operations.

Another attractive feature in the IBA network is the support for hardware-based multicast. Multicast is the ability to send a single message to a specific address and have it delivered to multiple processes which may be on different end nodes. This primitive is provided under the Unreliable Datagram (UD) transport mode, which is connectionless and unacknowledged. 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. Performance evaluations of this multicast primitive with the InfiniHost HCAs [8], InfiniScale switch and VAPI interface [9] show that it takes about 9.6μs to send a 1-byte message to 1 node and 9.8μs to send the message to 7 nodes. This shows that the operation is quite scalable and can be used effectively to design scalable collective operations.

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 utilized to our advantage. We propose three algorithms that utilize these features of IBA.

MVAPICH [12] is the implementation of Abstract Device Interface (ADI) [15] for the VAPI interface of the InfiniHost HCAs and is derived from MVICH [6] from Lawrence Berkeley National Laboratory. The algorithms for the barrier were implemented and integrated into the MVAPICH implementation of MPI over IBA, and we discuss the design and implementation issues here. We also present the results of our performance evaluations and show that considerable benefits are achieved using the proposed techniques.

2 Overview of RDMA and Multicast in InfiniBand

The InfiniBand Architecture (IBA) [3] defines a System Area Network (SAN) for interconnecting processing nodes and I/O nodes. It supports both channel and memory semantics. In channel semantics, send/receive operations are used for communication. In memory semantics, RDMA write and RDMA read operations are used instead of send and receive operations. These operations can directly access the memory address space of a remote process. They are one-sided and do not incur software overhead at the remote side.

InfiniBand provides hardware support for multicast. In some cases, this mechanism can greatly reduce communication traffic as well as latency and host overhead. InfiniBand also provides flexible mechanisms to manage multicast groups. However, multicast is only available for the Unreliable Datagram (UD) service. Therefore, tasks such as fragmentation, acknowledgment and retransmission, may be needed on top of UD to make multicast work reliably.

3 Barrier Algorithms

In this section we describe the three algorithms that we have designed and implemented for the barrier operation. 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. If the number of nodes performing the barrier is a power of two, then the number of steps in the algorithm is $\log_2 N$ and it is $\lfloor \log_2 N \rfloor + 2$ otherwise.

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, *barrier_id*. Each process has an array of bytes of length N. In each step of the PE, process i writes the *barrier_id* in the i th position of the array of the partner process j. It then waits for the *barrier_id* to appear in the j 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 1 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 *barrier_id*, 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.

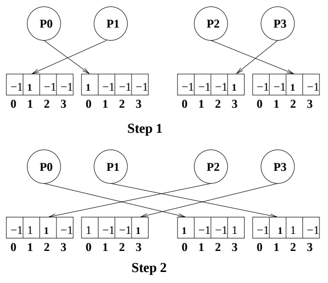


Fig. 1. Steps performed in RPE for a 4-node barrier

3.2 RDMA-based Dissemination (RDS)

In the 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 $j = (i + 2^m)\ mod\ N$. It then waits for a message from the process $k = (i + N - 2^m)\ mod\ N$. 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. More details on this algorithm are discussed in [4, 5].

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.

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 *barrier_id*. 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.

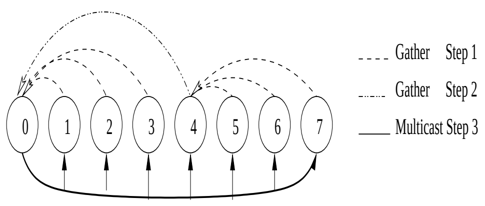


Fig. 2. Gather and Multicast Algorithm

Figure 2 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 algorithms.

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 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.

In our implementation, the buffers are allocated and registered during the first barrier call made by a process. This ensures that the memory is registered only if the application is involved in collective operations. Since the barrier is a collective call, during the first MPI_Barrier call, all the processes allocate memory for the barrier and perform the exchange of the virtual addresses. 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. Other options of registering buffers, including a dynamic scheme, are discussed in [4, 5].

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.

We make use of a scheme where the receiver polls 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. There is a static count called the *barrier_id* 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 *barrier_id*. 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.

4.3 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.

In our implementation each receiving process maintains a timer and sends a negative acknowledgment (NAK) when it has not received a message. When the root process receives this message, it retransmits the multicast message. Processes that have already received the message discard this retransmitted message.

The IB specification allows for event handlers to be executed 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.

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.

5 Performance Evaluation

We conducted our performance evaluations on the following two 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 InfiniBand 4x HCA [16]. 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.

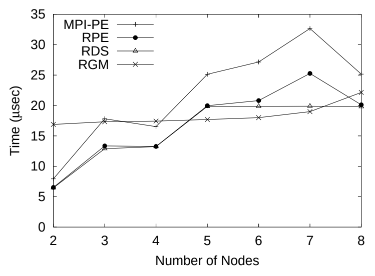
The barrier latency was obtained by executing MPI_Barrier 1000 times and the average of the latencies across all the nodes was calculated.

Figure 3 shows the performance comparisons of the three proposed barrier algorithms with MPI-PE, the standard pairwise exchange MPICH implementation of the barrier. We see that RPE and RDS perform better than MPI-PE for all cases. 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. 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. We see that for 8 nodes in Cluster 1, we gain as much as 1.25 factor of improvement with RPE and 1.27 with RDS. On 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.

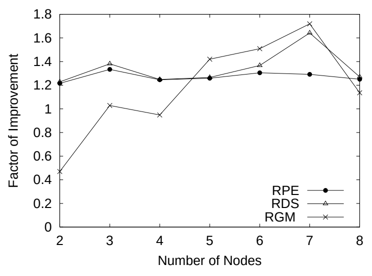
The factor of improvement for RPE is almost a constant in all cases because the benefit is obtained by the constant difference in the latency between a point-to-point send/receive operation and an RDMA-Write/poll operation.

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. Based on our experiments, we observed that a fan-in of 7 gives the best performance, and hence have chosen this value in our implementations.

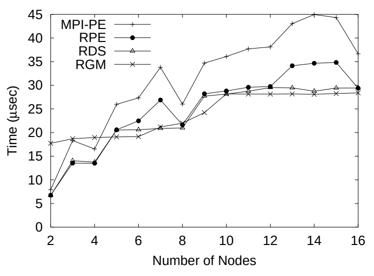
As mentioned earlier, the pairwise exchange algorithm does badly for non-power-of-2 group sizes because of 2 extra operations. Hence in order to do a fair comparison, we implemented the Dissemination algorithm with the point-to-point MPI functions. We refer to this as MPI-DS. The barrier latencies of the proposed algorithms are better than that of MPI-DS too. Figure 4 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.



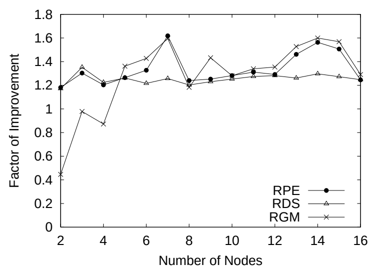
(a) **Latency (Cluster 1)**



(b) **Factor of Improvement**



(c) **Latency (Cluster 2)**



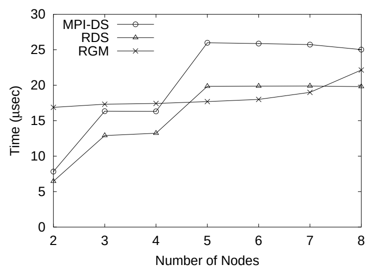
(d) **Factor of Improvement**

Fig. 3. Comparison of MPI-PE with the proposed algorithms for all group sizes on Clusters 1 and 2

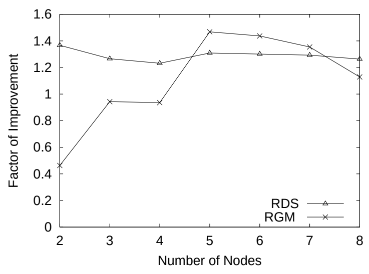
6 Related Work

The benefits of using RDMA for point-to-point message passing operations for IBA clusters has been described in [7]. 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.

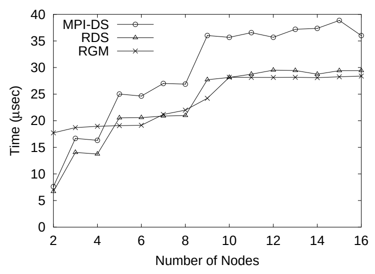
RDMA based design of collective operations for VIA based clusters [13, 14] has been studied earlier. Combining remote memory and intra-node shared memory for efficient collective operations on IBM SP has been presented in [17]. The implementations and performance evaluations of the barrier operation using remote mapped memory on the SCI interconnect was presented in [2]. However these papers do not focus on taking advantage of novel mechanisms in IBA to develop efficient collective operations.



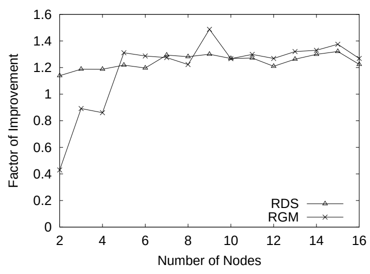
(a) **Latency (Cluster 1)**



(b) **Factor of Improvement**



(c) **Latency (Cluster 2)**



(d) **Factor of Improvement**

Fig. 4. Comparison of MPI-DS with the proposed algorithms on Clusters 1 and 2

7 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 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.

We are 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 and Jeff Kirk 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 from our research group for their help and valuable suggestions.

References

1. Mohammad Banikazemi, Rama K. Govindaraju, Robert Blackmore, and Dhabaleswar K. Panda. MPI-LAPI: An Efficeint Implementation of MPI for IBM RS/6000 SP Systems. *IEEE TPDS*, pages 1081–1093, October 2001.
2. Lars Paul Huse. Collective communication on dedicated clusters of workstations. In *PVM/MPI*, pages 469–476, 1999.
3. InfiniBand Trade Association. InfiniBand Architecture Specification, Release 1.0, October 24 2000.
4. Sushmitha P. Kini. Efficient Collective Communication using RDMA and Multicast Operations for InfiniBand-Based Clusters. Master Thesis, The Ohio State University, June 2003.
5. Sushmitha P. Kini, Jiuxing Liu, Jiesheng Wu, Pete Wyckoff, and Dhabaleswar K. Panda. Fast and Scalable Barrier using RDMA and Multicast Mechanisms for InfiniBand-Based Clusters. Technical Report, OSU-CISRC-05/03-TR24, May 2003.
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, Jiesheng Wu, Sushmitha P. Kini, Pete Wyckoff, and Dhabaleswar K. Panda. High Performance RDMA-Based MPI Implementation over InfiniBand. In *ICS '03*, June 2003.
8. Mellanox Technologies. Mellanox InfiniBand InfiniHost Adapters, July 2002.
9. Mellanox Technologies. Mellanox IB-Verbs API (VAPI), Rev. 0.97, May 2003.
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. Network-Based Computing Laboratory. MVAPICH: MPI for InfiniBand on VAPI Layer. http://nowlab.cis.ohio-state.edu/projects/mpi-iba/index.html, January 2003.
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. 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.
16. Topspin Communications, Inc. Topspin InfiniBand Host Channel Adapter, http://www.topspin.com/solutions/hca.html.
17. V. Tipparaju, J. Nieplocha, D. K. Panda. Fast Collective Operations Using Shared and Remote Memory Access Protocols on Clusters. In *IPDPS '03*, April 2003.