Efficient Barrier using Remote Memory Operations on VIA-Based Clusters*

Rinku Gupta* Vinod Tipparaju † Jarek Nieplocha† Dhabaleswar Panda*

*The Ohio State University
{guptar, panda}@cis.ohio-state.edu

†Pacific Northwest National Laboratory
{vinod.tipparaju, jarek.nieplocha}@pnl.gov

Abstract

Most high performance scientific applications require efficient support for collective communication. Point-to-point message-passing communication in current generation clusters are based on Send/Recv communication model. Collective communication operations built on top of such point-to-point message-passing operations might achieve suboptimal performance. VIA and the emerging InfiniBand architecture support remote DMA operations. Such operations not only allow data to be moved between the nodes with low overhead, they also allow to create and provide a logical shared memory address space across the nodes. This feature is valuable for implementing high performance and scalable collective operations. In this paper, we focus on barrier, probably the most frequently-used collective operation. We demonstrate how RDMA write operations can be used to support inter-node barrier in a cluster with SMP nodes. We also propose a scheme to exploit the shared memory within a SMP node to support fast intra-node barrier. Combining these two schemes, we developed an overall barrier scheme for cluster of SMP nodes with cLAN VIA interconnect. Compared to the current barrier algorithms using send/recv communication model, the new approach is shown to reduce barrier latency on a 64 processor (32 dual nodes) system by up to 66%. These results demonstrate that high performance and scalable barrier implementations can be delivered on current and next generation VIA/Infiniband-based clusters with RDMA support.

Keywords: *Remote DMA, Virtual Interface Architecture, Collective Operations, Barrier, and Interprocessor Communication*

1 Introduction

Cluster computing systems are becoming increasingly popular for providing *cost-effective* and *affordable* computing environments for day-to-day computational needs of a wide-range of applications. These systems are typically built by interconnecting a set of commodity PCs/workstations with commodity *Local Area Networking* (LAN)/ *Systems Area Networking* (SAN) technologies (such as Fast Ethernet [15], Gigabit Ethernet[8], Myrinet [5], ATM [2], and GigaNet [1]).

Distributed and high performance scientific applications running on such clusters require efficient support for both *point-to-point* and *collective* communication [16]. Frequently-used collective communication operations include: barrier, broadcast, all-reduce, gather, scatter, etc. Many times collective operations fall in the critical path of a program's execution. Thus, providing high-performance and scalable support for collective communication operations is critical for any cluster.

The increasing performance and availability of Local Area Networks (LANs)/ System Area Networks (SANs) is shifting the communication bottlenecks in clusters from network fabrics to communication overheads at the senders and receivers. In clusters with the traditional communication architecture, sending a message from one node to another used to involve multiple copies and context switches to the kernel at the sender and the receiver side. With the advent of user-level communication protocols such as AM[24], FM[18], U-Net[23], LAPI[20], EMP[21], these multiple copies and the kernel context switch have been purged from the critical path the message takes.

Communication in most current generation message passing protocols adheres to the the Send/Recv model. Both the sender and the receiver actively participate in the communication step. Generally in user level protocols, the Send/Recv model requires explicit posting of descriptors on both the Send and Receive side and then sending the data.

Modern user level protocols such as Virtual Interface Architecture (VIA) [11] offer another method for communication namely the Remote memory operation or Remote DMA (RDMA). The concept of Remote DMA transfer is used for direct transfer of data between user spaces. Remote DMA write allows the sender to specify the data storage area on the receive side, and then directly write data to that area. Remote DMA Read allows the receiver

*This Research is supported in part by Department of Energy's Grant # DE-FC02-01ER25506 and an NSF Grant #EIA-9986052.

to specify the address on the sender side and directly read data from that location. The RDMA facility is very much like the get-put model supported by any higher-level libraries like ARMCI [17]. Using RDMA relieves the receiver from the task of posting descriptor, or allocating buffers before a send takes place. This is because RDMA in itself is transparent to the receiver. The sender knows the memory address of the destination node and can directly write to it.

This raises an interesting challenge whether this remote memory operation capability can be used to support efficient collective communication in clusters. Past works in the collective communication area have primarily focused on development of optimized and scalable algorithms on top of point-to-point operations [13, 4, 19]. These point-to-point operations are typically supported by Send/Recv model of communication. Remote memory capability through RDMA operations allows to define a set of buffers across the nodes of a cluster which can be used as a logical shared address space to exchange data efficiently. As modern clusters are being built with Symmetric Multi-Processor (SMP) nodes, it also provides shared memory within a node to support efficient intra-node communication. This raises the following open questions:

1. Can remote operations be used to support efficient inter-node communication steps for a collective operation?
2. Can shared memory communication within an SMP node be used to support efficient intra-node communication steps for a collective operation?
3. How both these features can be combined to support high performance and scalable collective operations on modern and next generation clusters consisting of SMP nodes?

In this paper, we take on such a challenge. We focus on barrier synchronization, a frequently-used collective communication operation. First, we show how remote memory operations (such as RDMA write) can be used to support efficient inter-node barrier on an SMP cluster. Various design issues and alternatives associated with this approach are presented and analyzed. Next, we describe our approach of exploiting shared memory within an SMP node to support efficient intra-node barrier. Combining both these solutions lead to a high performance implementation of barrier for cluster of SMP nodes.

Compared to the current barrier algorithm in MVICH/MPICH using send/recv communication model, our new approach is shown to reduce barrier latency on a 64 processor (32 dual nodes) system by up to 66%.

The paper is organized as follows. Section 2 provides an overview of Virtual Interface Architecture and Message Passing Interface(MPI) Section 3 presents the basic idea behind using remote operations. Design issues are presented in Section 4. Section 5 describes our new Barrier algorithm. Design solutions and implementation details are presented in Section 5. Experimental results are presented in Section 6. Conclusions and future work are presented in Section 7.

2 Overview of VIA and MPICH/MVICH

In this section, we provide a brief overview of VIA and MPICH/MVICH architecture.

2.1 Virtual Interface Architecture

The Virtual Interface Architecture (VIA) has been proposed as a standard for user-level communication protocols for low latency and high bandwidth SANs. The VIA specification mainly aims at reducing the number of copies associated with a message transfer and removing the kernel from the critical path the message takes. It is achieved by providing user applications a protected and directly accessible network interface called a Virtual Interface (VI).

Figure 1 describes the Virtual Interface Architecture. Each VI is a communication endpoint consisting of a Send Queue and a Receive Queue. Applications post requests to these queues in the form of VIA descriptors. Each descriptor contains a Control Segment (CS), zero or more Data Segments (DS) and possibly an Address Segment (AS). Each DS contains a user buffer virtual address. The descriptor gives necessary information including the data buffer address and length. VIA requires that the memory buffers used in the data transfer to be registered to avoid swapping out of the pages and transferring data directly from the buffers to the network. For each contiguous region of memory registered, the application (VI consumer) gets an opaque handle. The registered memory can be referenced by the virtual address and the associated memory handle.

The VIA specifies two types of data transfer facilities: the traditional send and receive messaging model and the Remote Direct Memory Access (RDMA) model. The VI Descriptor distinguishes between the Send and RDMA. The Send descriptor contains the CS and DS. In case of RDMA, the VI descriptor also contains the AS. In the AS, the user specifies the address of the buffer at the destination node and the memory handle associated with that registered destination buffer address.

In the send and receive model, each send descriptor on the local node has to be matched with a receive descriptor on the remote node. Failure to post a receive descriptor on the remote node results in the message being dropped and if the connection is a reliable connection, it even results in the breaking of the connection.

In the RDMA model, the initiator specifies both the virtual address of the local user buffer and that of the remote

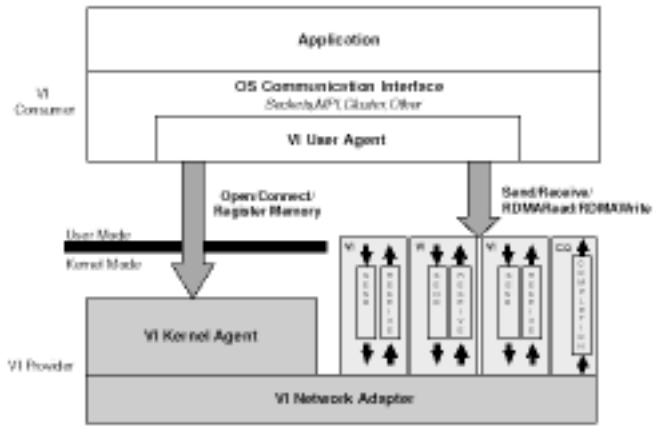


Figure 1: VI Architectural Model

user buffer. In this model, a descriptor does not have to be posted on the receiver side corresponding to every message. The exception to this case is when the RDMA write is used in conjunction with immediate data, a receive descriptor is consumed at the receiver end.

There are two types of RDMA operations: RDMA write and RDMA read. In the RDMA write operation, the initiator specifies both the virtual address of the locally registered source user buffer and that of the remote destination user buffer. In the RDMA Read operation, the initiator specifies the source of the data transfer at the remote and the destination of the data transfer within a locally registered contiguous memory location. In both cases, the initiator should know the remote address and should have the memory handle for that address beforehand.

Since the introduction of VIA, many software, firmware and hardware implementations of VIA have become available. The Berkeley VIA [6], Firm VIA [3], M-VIA [14], ServerNet VIA [22], GigaNet VIA [12] are among these implementations. In this paper, we focus on the RDMA write feature of VIA and use GigaNet VIA for experimental evaluation.

2.2 MPICH/MVICH

Message Passing Interface [16] is a standard library specification for message passing. MPI includes point-to-point message passing and collective communication, all scoped to a user specified group of processes. MPI provides abstractions for processes at two levels. First, processes are named according to the rank of the group in which the communication is being performed. Second, virtual topologies allow graph or Cartesian naming of processes that help relate the application semantics to the message passing semantics in a convenient and efficient way. A key concept in MPI is that of a communicator, which provides a safe message-passing context for the multiple layers of software within an application that may need to perform message passing. For example, messages from a support library will not interfere with the other messages in the application, provided the support library uses a separate communicator. Communicators, which house groups and communication context (scope) information, provide an important measure of safety that is necessary and useful for building up library-oriented parallel code. Within a communicator, point-to-point and collective operations are also independent.

MPICH [9], which combines portability with high performance is one of the most popular implementations of Message Passing Interface. A new version of MPICH, called MVICH [10], was developed over the VIA. We evaluate our scheme using MVICH 1.0.

3 Basic Concept

In a shared memory system, a barrier operation can be done very easily. A specific memory location can be reserved for the barrier operation and initialized to '0'. When processes reach the barrier, they can simply increment the value at the location (in an atomic manner) and then wait for the value to be updated by all other processes.

Another approach is to have a section of memory (with multiple locations) be reserved for barrier and initialized to '0'. Every process can write a '1' to a specified location in this memory region when it reaches the barrier. Next, it reads from other memory locations to see if other nodes have reached the barrier. This concept is illustrated in Figure 2 with four processes (P0, P1, P2 and P3) and four memory locations. The figure shows the memory location corresponding to each process. In this figure, P0, P2, and P3 have already set the byte in the respective locations when they encounter the barrier and are waiting for P1 to set the value in its location. As soon as P1 sets the value in its location, all processes view it and can be released from their barrier.

Efficient execution of the above shared memory-based barriers require several issues related to cache coherency to be addressed. If the shared memory is cache coherent, the barrier implementation turns out to be considerably simpler and faster. The processes obtain the data by a simple local read operation without additional complexities. However to reduce false sharing, the memory locations associated with the processes need to be mapped to different cache lines to eliminate false sharing [7].

In a cluster with distributed memory organization, when

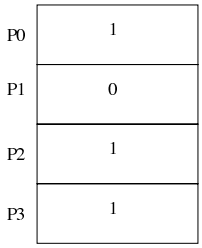


Figure 2: Illustration of a simple barrier scheme using shared memory

an operation like barrier takes place, the nodes typically send and receive explicit messages. Some kind of barrier algorithms (pair-wise exchange with recursive doubling or gather-followed-by-broadcast) with multiple phases (steps) are used to implement the barrier. Each of the communication steps typically use a send/recv model to communicate. Receiving a message from a node is typically an expensive operation in higher level libraries over user level protocols like VIA. For example, in the MPI library, a receiver has to take care of unexpected receive messages. When messages come in, the relevant descriptor has to be searched for. If there is no descriptor posted, data is sent to an intermediate buffer. When the actual descriptor gets posted, the data has to be copied from the temporary buffer to the user buffer. In addition, the layering structure of the middle libraries adds considerable overhead on the message latency, making each of the communication step slower and the entire barrier operation slower.

The method of RDMA communication offers a new mechanism for transferring data, by directly writing into the memory of a remote processor/node. Consider a set of buffers being allocated at each remote processor/node and their addresses being exchanged before execution of a program. The collection of these buffers (together with their addresses) provide a logical shared memory region (without coherency) for all processors. Now, the processors can exploit the advantages associated with shared memory-based algorithms to implement barrier. Since RDMA operations are typically faster than send/recv communication, this approach also promises better performance. In addition, posting receive descriptor is not required for RDMA operations to complete.

Modern clusters consist of network of SMP nodes where each node can have multiple processors. We exploit the coherent shared memory within a node to support efficient intra-node barrier steps. We exploit the logical shared memory capability provided by RDMA operations to support efficient inter-node barrier steps. A combination of these two approaches lead us to a better barrier implementation in MPI compared to the current implementation using send/recv communication steps.

In the next sections, we discuss the major design issues, alternative solutions and their trade-offs and describe the implementation details.

4 Design Issues

The idea behind using RDMA for barrier is to utilize the concepts of shared memory. The RDMA mechanism and memory registration constraints open up several major issues for designing an RDMA based barrier. One issue is how and when to register the buffers and communicate the addresses of individual buffers to all nodes. Another issue is how to identify valid data at the receiver.

In this section, we discuss these design issues and present some solutions. In Section 6 we discuss the design choices for our implementation.

4.1 Registration of buffers and Address Exchange

It is a requirement in VIA that data to be sent and received should only be from and to registered buffers. A flexible buffer management scheme is required for this purpose in context of collective operations. Keeping this in consideration, one solution is to statically register contiguous buffers in memory for each communicator when the communicator is created. In this approach the address of only the starting buffer needs to be communicated to all nodes as the length of the buffer space is the same for all nodes in the same communicator. There will be certain constraints to the order of using these buffers, which we shall discuss in the Section 6. Also, because this is a static scheme the address is communicated only once and the buffers are to be reused as and when needed.

Another approach would be to allow dynamic registration and use of non-contiguous buffers. This will make it mandatory to communicate the addresses of all the buffers to all the nodes. Dynamic registration need not be done in the start of the program but can be done as and when needed. However, the disadvantage of this approach is that the buffer addresses need to be communicated whenever the buffers are created dynamically. Hence, if we register the buffer in the collective operation, we will have additional overhead of address communication with the destination set in the collective operation before sending the actual data to the destination set.

4.2 Data Validity at the Receiver end

No data is communicated to the receiver in the barrier operation. RDMA write is receiver transparent. It does not require that the receiver posts a descriptor or performs any action in anticipation of the incoming data. The receiver process receives no indication that a new data has been written. When the destination needs the data it goes to the memory location and fetches the data from there. We need a mechanism for indicating to the receiver that the data in the memory is valid data.

One method is to let the receiver NIC interrupt the receiver once it receives an RDMA message but this is a

very expensive operation and thus not helpful to achive high performance.

Another approach is to use the immediate field in the RDMA descriptor, and set the field when the last RDMA write operation has taken place. However, this requires a consumption of a descriptor at the receive end. This also requires that the receiver be aware of the data coming and post a receive descriptor in advance. This approach disturbs the illusion of shared memory and is not feasible.

Another approach is to let the sender write a special data value, which will be known by the receiver in advance. The presence of that special value will indicate to the receiver that data has been written to its memory location by the sender.

5 Our Algorithm

In this section, we introduce our intra-node, inter-node and overall barrier algorithms.

5.1 Inter-node Barrier

The algorithm we use is pairwise exchange with recursive doubling. This algorithm was chosen for its simplicity and efficiency. This algorithm is also currently used in MVICH 1.0 distribution with Send/Recv communication model. It also enables us to give us a fair comparison between efficiency of the Barrier using Send/Recv with our new implementation of Barrier with RDMA.

When the RDMA barrier is called, the pairwise exchange method follows. During each barrier involving the same communicator, each process keeps a static count of the barrier number it is participating in that communicator. The Pairwise Exchange Algorithm is a recursive algorithm. The nodes pair up and each node does an RDMA write to the other process buffer using the destination buffer address and memory handle. The sender writes the barrier number which it is involved in for that communicator. Since the receiver is also in the same barrier for that communicator, it also knows the barrier number and it can read the barrier number from its location. Thus, the nodes in the pair do a RDMA write to each others memory and read the data that the other node has written from its own local memory. The nodes form a group. In the next step two groups pair up and a node from one group does an RDMA write and checking for written data with one node from the other group. These groups are then merged together. This process of pairing up, writing data to each others buffers and then merging is repeated until only one group is left. The barrier is then finished. Overall, in this approach each node performs log_2N RDMA writes, where N is the number of nodes in power of two.

For non-power of two nodes, we divide the set of nodes N into two sets S and S' where S is the maximum power of two less than N and S' is the set of nodes in N but not in S. Initially, every node in S' does an RDMA write to another node in S. Then the nodes in S perform a pairwise exchange barrier. Once the nodes in S reach a barrier, they RDMA write to the corresponding nodes in S'. This concludes the barrier for the N nodes. The number of steps it takes to reach the barrier is $log_2N + 2$ steps.

5.2 Intra-node Barrier

A barrier algorithm within an SMP node is implemented using a flat tree. For small and medium size of the SMP nodes such as in the commodity Linux clusters, we found this approach fast and sufficiently scalable. The algorithm is very simple and involves using one flag per process. That flag is located in shared memory and we assure that each flag is located on a different cache line. That flag is set by the corresponding process to indicate that the process arrived at a barrier. After that the process waits until that flag is reset which indicates that all the processes on the node were synchronized. One process on an SMP node is selected as a master. Master process waits for all processes to check in and then it resets the value of the flag for all other processes.

5.3 Overall Barrier

The above two algorithms are merged for implementing barrier on a cluster of SMP nodes. First the master process within an SMP node waits until all processes on the node check in at the barrier, then it executes inter-node barrier algorithm involving one master process on each node, and then it resets the value of all flags to notify the other processes that the global barrier operation is complete.

6 Design Solutions and Implementation Details

We currently restrict ourselves to the RDMA Write operation because RDMA-Read is an optional feature of the VIA specification and not all implementations of VIA support it. Now we discuss the design solutions for the implementation of the inter-node RDMA Barrier with GigaNet cLAN as the cluster interconnect.

We indicate the nodes that take part in the barrier by the term 'barrier group'. As mentioned above, for cluster of SMP nodes, the master nodes form the barrier group.

6.1 Buffer Registration

We register a buffer with every process.For SMP nodes, we register the buffer for only the master process, of size

equal to the number of nodes in the barrier group. Every byte in the buffer is reserved for a different node in the barrier group. Nodes are differentiated on the basis of their id in their barrier group. Each node has a different id in the barrier group. The same buffer is reused for all the other following barrier operations and no new buffers need to be registered.

The first byte is reserved for the node with id 0, the second byte for the node with id 1 and so on. Figure 3 shows inter-node RDMA Barrier in a 4 node cluster having processes P0, P1, P2, P3 each on a different node. Each node has reserved a four byte buffer, with one byte reserved for every node in that barrier group.

6.2 Address exchange

For a node to write data in some other process's memory, the node needs to know the destination address and have the right memory handle. Thus, address of the buffers need to be communicated from one node to all the other nodes. Since the buffers are contiguously allocated and have the same handle, only the start address needs to be communicated to the other nodes. The implementation does address exchange when the barrier group is created using explicit Send-Recv primitives. Communicating the address need to be done only once.

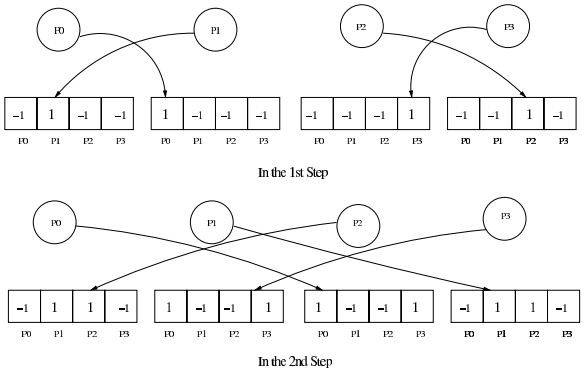


Figure 3: Inter-node RDMA Barrier between 4 nodes

6.3 Buffer Initialization

Since barrier is essentially a synchronization operation and the data passed is not relevant; we initialize all the buffer bytes reserved for barrier operation to a negative constant. It shall be shown in the next sub-section that we never indicate a barrier operation by a negative number and hence the initial value of negative constant suffices. Figure 3 shows the first barrier operation of a program. The buffer bytes are initialized to the negative constant -1.

6.4 Data identification at the Receiver end

When the receiver needs the data, it goes to the corresponding memory location and reads the data from there. The algorithm writes the barrier number at the destination. The receiver knows the barrier number that can be written. Hence it checks the location for the desired data. However the implementation does not check for the desired value, but instead it checks to see if the value written in its location by the other node is greater than or equal to the barrier number. This is to take care of consecutive barriers. In a barrier between a pair of nodes, one of the participating nodes may be slower than the other. The faster node may enter the second barrier before the slower exists the first one. Thus the faster node may overwrite the barrier number by the barrier number of the next barrier. Thus, as shown in Figure 3 if the nodes in the cluster are in the first barrier, process P0 will poll for a value greater than 1 to be written in its buffer by the various processes.

7 Experimental Results

In this section, we discuss the results that have been obtained for RDMA Barrier and compare it with the results for the MPI Barrier for a cluster of uniprocessor and SMP nodes.

We evaluated out implementation on the following clusters.

Cluster 1: A cluster of 8 nodes, each with a 66MHz PCI bus, 700MHz Pentium III machines, 1GB of Main memory and Linux version 2.2.17. The machines are connected using a GigaNet 5300 switch.

Cluster 2: A cluster of 16 nodes, each with a 33MHz PCI bus, 1000MHz Pentium III machines, 512MB of Main memory and Linux version 2.2.17. The machines are connected using a GigaNet 5300 switch.

Cluster 3: A cluster of 32 dual 500 MHz PIII machines. The nodes are PowerEdge-1300. The machines are connected using GigaNet 5000 and 5300 switches.

To obtain the barrier latency, we ran 10000 iterations of MPI_Barrier and took the average of the barrier latencies at each node. The MVICH version used is mvich-1.0.

7.1 Inter-Node Barrier Evaluation

We ran the original MPI_Barrier without modification, the results of which are labeled under Message Passing. The inter-node RDMA Barrier is labeled under RDMA.

Figures 4(a) and 4(b) show the barrier latencies for power of 2 nodes for Cluster 1 and Cluster 2 respectively.

For Cluster 1, the RDMA Barrier for 8 nodes completes in 31.88μs as compared to the Message Passing Barrier,

which completes in 45.14μs. For every message sent, we save 4μs in one-way latency. The RDMA Barrier outperforms the Message Passing Barrier for all power of 2 cases. This leads to 29.37% performance improvement

Similar results are obtained in Cluster 2 where we see that RDMA Barrier for 8 nodes completes in 29μs as compared to 40.5μs of the Message Passing Barrier. The results for 16 nodes in Cluster 2 show that RDMA Barrier completes in 39.4μs as compared to Message Passing Barrier which takes 56.3μs. This leads to 28.4% improvement on the 8 nodes and 30% for 16 nodes and is thus scalable.

We also obtained the inter-node barrier latency for power of two nodes on Cluster 3. Evaluation of the inter-node barrier is done by running one task on each node of Cluster 3. In Figure 4(c) we obtain an improvement up to 41% for 16 and 32 nodes using RDMA Barrier.

Figures 5(a) and 5(b) show the barrier latency for all nodes in Cluster 1 and Cluster 2. The barrier latency for non-power of 2 nodes is greater than the power of 2 nodes because they execute larger number of steps. The timings for non power of two nodes also demonstrate an improvement in performance of inter-node RDMA Barrier as compared to Message Passing Barrier.

7.2 Overall Barrier Evaluation

The overall barrier was evaluated on Cluster 3, which contains dual SMP processors. Our barrier uses shared memory protocol within SMP node and RDMA accross the network.

Evaluation of the overall barrier by using both the dual processors on each node, by running two tasks on each node. Figure 6 shows graphs for the overall barrier algorithm that integrates shared memory and RDMA, and the corresponding MPI barrier operation that is based on point-to-point message passing.

We observe increased performance advantage of the RDMA barrier over the MPI barrier as the number of processors grows. Our barrier is able to exploit shared memory very effectively as the difference between one and two task per node results is very close. In case of the MVICH barrier, the gap between these two configurations is substantial and growing as the number of nodes increases. Overall, the proposed RDMA barrier outperforms the barrier based on point-to-point message passing by a large factor. For example, on 32 nodes and 64 tasks the message passing barrier takes 193μs whereas the overall RDMA barrier takes only 65μs. We see an performance improvement of 66%.

8 Conclusion and Future Work

We have presented a new approach for implementing efficient barrier on clusters with SMP nodes. This scheme exploits remote memory operations across nodes and shared

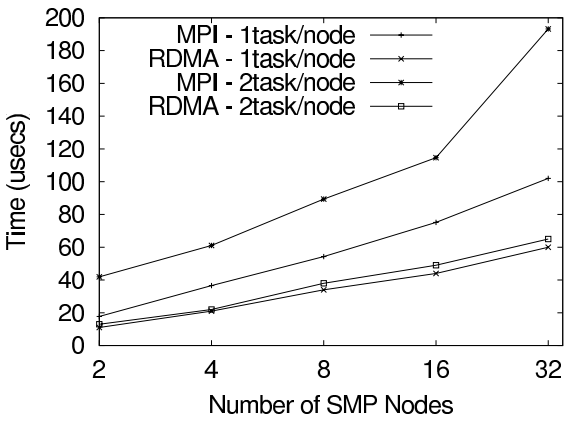


Figure 6: Performance of MPI and RDMA barrier on cluster with 32 dual CPU nodes using one and two tasks per node

memory within an SMP node. The barrier algorithms together with design and implementation issues on clusters with GigaNet cLAN VIA are presented. The complete implementation is evaluated on three different cluster configurations. The results demonstrate that the new scheme delivers up to 66% reduction in barrier latency on a 64 processor (32 dual processor nodes) cluster. The results also demonstrate that the proposed scheme is scalable and can deliver better performance as the system size increases.

In the current paper, we have focused on barrier. We are extending our framework for other common collective operations such as broadcast, and all-reduce operations. We plan to incorporate the new schemes into the collective communication libraries for VIA/InfiniBand clusters. We also plan to study the application-level performance benefits of the proposed new schemes.

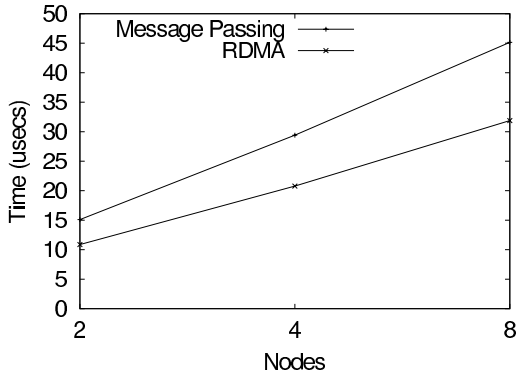
References

[1] GigaNet Corporations. http://www.giganet.com.

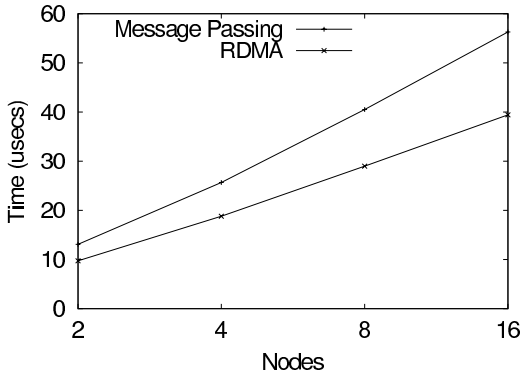
[2] ATM Forum. *ATM User-Network Interface Specification, Version 3.1*, September 1994.

[3] M. Banikazemi, V. Moorthy, L. Hereger, D. K. Panda, and B. Abali. Efficient Virtual Interface Architecture Support for IBM SP switch-connected NT clusters. In *the Proceedings of the International Parallel and Distributed Processing Symposium, pages 33-42*, May 2000.

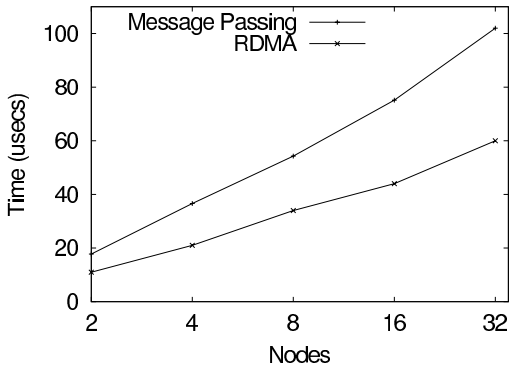
[4] M. Barnett, R. Littlefield, D. G. Payne, and R. Van de Geijn. Global Combine on Mesh Architectures with Wormhole Routing. In *Proceedings of the International Parallel Processing Symposium*, pages 156–162, 1993.



(a) Cluster 1

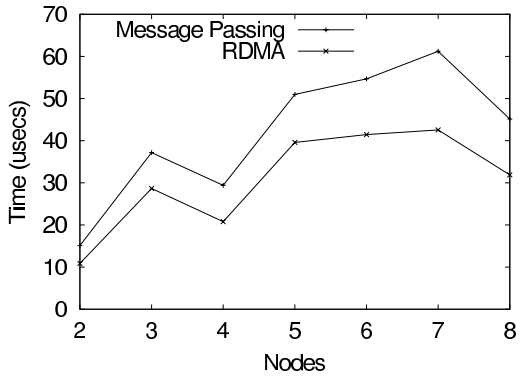


(b) Cluster 2

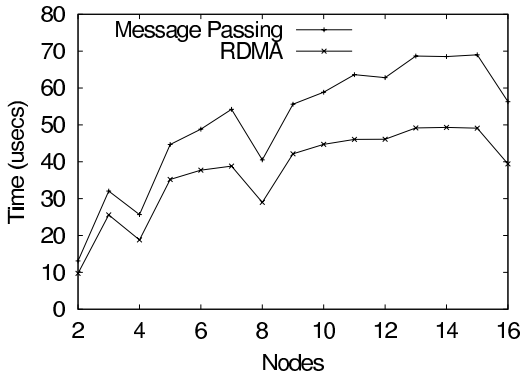


(c) Cluster 3

Figure 4: Barrier Latency for Power of two nodes



(a) Cluster 1



(b) Cluster 2

Figure 5: Barrier Latency for All nodes

[5] N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic, and W. K. Su. Myrinet: A Gigabit-per-Second Local Area Network.

[6] P. Buonadonna, A. Geweke, and D. E. Culler. BVIA: An Implementation and Analysis of Virtual Interface Architecture. In *the Proceedings of Supercomputing '98*, 1998.

[7] D. E. Culler and J. P. Singh. *Parallel Computer Architecture: A Hardware-Software Approach*. Morgan Kaufmann, March 1998.

[8] H. Frazier and H. Johnson. Gigabit Ethernet: From 100 to 1000Mbps.

[9] W. Gropp and E. Lusk. MPICH Working Note: The Second Generation ADI for the MPICH Implementation of MPI.

[10] Future Technology Group. MVICH: MPI for Virtual Interface Architecture. In *http://www.nersc.gov/research/FTG/mvich*.

[11] http://www.viarch.org/. Virtual Interface Architecture Specifications.

[12] GigaNet Incorporations. cLAN for Linux: Software Users' Guide. 2001.

[13] S. L. Johnsson and C.-T. Ho. Optimum Broadcasting and Personalized Communication in Hypercubes. *IEEE Transactions on Computers*, pages 1249–1268, September 1989.

[14] M-VIA: A High Performance Modular VIA for Linux. http://www.nersc.gov/ research/FTG/via.

[15] Lee Melatti. Fast Ethernet: 100 Mbit/s Made Easy. *Data Communications on the Web*, Nov. http://www.data.com/tutorials/100mbits-made-easy.html.

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

[17] J. Nieplocha and B. Carpenter. ARMCI: A Portable Remote Memory Copy Library for Distributed Array Libraries and Compiler Run-time Systems. In *3rd Workshop on Runtime Systems for Parallel Programming (RTSPP) of International Parallel Processing Symposium IPPS/SPDP '99*, April 1999.

[18] S. Pakin, M. Lauria, and A. Chein. High Performance Messaging on Workstations: Illinois Fast Messages (FM). In *Proceedings of SC*, 1995.

[19] D. K. Panda. Global Reduction in Wormhole k-ary n-cube Networks with Multidestination Exchange Worms. In *International Parallel Processing Symposium*, pages 652–659, Apr 1995.

[20] G. Shah, J. Nieplocha, J. Mirza, C. Kim, R. Harrison, R. K. Govindaraju, K. Gildea, P. DiNicola, and C. Bender. Performance and Experience with LAPI: A New High Performance Communication Library for the IBM RS/6000 SP. In *the Proceedings of the International Parallel Processing Symposium '98*, March 1998.

[21] P. Shivam, P. Wyckoff, and D. Panda. EMP: Zero-copy OS-bypass NIC-driven Gigabit Ethernet Message Passing. In *the Proceedings of Supercomputing '01*, November 2001.

[22] E. Speight, H. Abdel-Shafi, and J. K. Bennett. Realizing the Performance Potential of the Virtual Interface Architecture. In *Proceedings of the International Conference on Supercomputing*, June 1999.

[23] T. von Eicken, A. Basu, V. Buch, and W. Vogels. U-Net: A user-level network interface for Parallel and Distributed Computing. In *the Proceedings of the 15th ACM Symposium on Operating Systems Principles*, December 1995.

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