Efficient Collective Operations using Remote Memory Operations on VIA-Based Clusters[*]

Rinku Gupta[†] Pavan Balaji[†] Dhabaleswar K. Panda[†] Jarek Nieplocha[‡]

[†]The Ohio State University
{guptar, balaji, panda}@cis.ohio-state.edu

[‡]Pacific Northwest National Laboratory
jarek.nieplocha@pnl.gov

Abstract

High performance scientific applications require efficient and fast collective communication operations. Most collective communication operations have been built on top of point-to-point send/receive primitives. Modern user-level protocols such as VIA and the emerging InfiniBand architecture support remote DMA operations. These operations not only allow data to be moved between the nodes with low overhead but also allow the user to create and provide a logical shared memory address space across the nodes. This feature demonstrates potential for designing high performance and scalable collective operations. In this paper, we discuss the various design issues that may be the basis of a RDMA supported collective communication library. As a proof of concept, we have designed and implemented the RDMA-based broadcast and the RDMA-based allreduce operations. For RDMA-based broadcast, we get a benefit of 14%, when compared to send/receive-based broadcast for 4KB data size on a 16 node cluster. We also introduce a new reduce algorithm called as the Degree-k tree-based reduce algorithm. Combining the RDMA mechanism with the new reduce algorithm shows a benefit of 38% for 4 byte messages and 9% for 4KB messages on a 16 node cluster for the allreduce operation. We also introduce analytical models for broadcast and allreduce to predict the performance of this design for large-scale clusters. These analytical models yield a performance benefit of about 35-40% for 4 bytes and around 14% for 4KB messages for 512 and 1024 node clusters for the allreduce operation.

1 Introduction

High Speed interconnection networks and exponentially increasing microprocessor performance have made Networks of Workstations (NOWs) an increasingly appealing alternative to mainstream supercomputing for a variety of computational needs of computation intensive applications. Commonly known as Cluster Computing systems, these collections of commodity based components offer a high performance to price ratio to the end user, attributing to it's immense success. High Performance Parallel Programs on clusters often involve a lot of point-to-point and collective communication between them in addition to the computation being carried out.

Past works in the collective communication area have primarily focused on development of optimized and scalable algorithms on top of point-to-point send/receive operations [3]. The send/receive model requires explicit host intervention at both the sender and the receiver side. Modern user-level protocols such as the Virtual Interface Architecture (VIA) [8] and the InfiniBand Architecture (IBA) [1] offer a variety of models for data transfer. Together with the send/receive model, they also support the Remote Direct Memory Access (RDMA) model. The concept of Remote DMA is used for direct transfer of data between user spaces without any intervention from the receiving host. In other words, the RDMA operation is transparent to the receiver. Remote memory capability through RDMA operations allows the programmer 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. This raises the following open question. Can remote memory operations be used to design and implement efficient collective communication operations?

In our earlier work, we provided RDMA support for implementing fast barrier synchronization [7]. In this paper, we take up the challenge of exploring ways to design data-intensive collective operations such as broadcast and allreduce using the RDMA mechanism. We analyze various design issues and alternatives for supporting such collective operations with RDMA supported shared memory. For the broadcast and allreduce operations, we demonstrate how these issues have been resolved in practice in the design of high-performance collective communication libraries.

We introduce a new reduce algorithm called the *Degree-k* tree-based reduce algorithm. The implementation of allreduce using this new algorithm along with the RDMA mechanism gives significant performance benefits compared to the traditional send/receive-based allreduce operation. This benefit was found to be 38% and 9% for small (4 bytes) and large (*4KB*) messages respectively on a 16 node GigaNet cLAN cluster. To allow MPI applications take advantage of the new implementation, we linked the RDMA-based broadcast and allreduce algorithms with MVICH (a

[*]The OSU part of this Research is supported in part by Department of Energy's Grant #DE-FC02-01ER25506, and National Science Foundation's grants #EIA-9986052 and #CCR-0204429.

popular MPI implementation for VIA) [4].

We also present analytical models to find the optimal RDMA-based allreduce algorithm for a given configuration and data size, and to estimate the performance of the RDMA-based broadcast operation for a given data size. We use this to predict the performance benefits of using the RDMA-based collective operations for large clusters. The analytical model predicts a 20% improvement in the broadcast latency for 512-node systems. The predicted performance for RDMA-based allreduce shows a benefit of about 35-40% for small messages of 4 bytes and around 14% for messages of *4KB* size for 512 and 1024 node clusters. These results demonstrate that efficient collective operations can be built on next generation clusters with networks (such as InfiniBand and Quadrics) supporting RDMA-based mechanisms.

The remaining part of the paper is organized as follows. Section 2 provides an overview of VIA [8]. In Section 3, we discuss the basic design issues. The RDMA-based broadcast and allreduce, along with their design issues are discussed in Section 4. Section 5 provides the analytical models for broadcast and allreduce. We present the performance results (experimental and analytical) in Section 6 and conclude the paper in Section 7.

2 Overview of Virtual Interface Architecture

The Virtual Interface Architecture (VIA) has been standardized as a low latency and high bandwidth user-level protocol for System Area Networks (SANs). VIA provides every consumer process a protected and directly accessible interface to the network named as a Virtual Interface (VI).

Each VI is a communication endpoint, consisting of send and receive queues. Applications post requests to the send and receive queues in the form of VI send and receive descriptors. VIA requires that buffers used in the communication be registered/pinned. The registered buffer address is communicated using the VI descriptor. VIA specifies two types of data transfer facilities: the Send/Receive messaging model and the Remote Direct Memory Access (RDMA) model. In the Send/Receive model, each send descriptor on the local node has to be matched with a receive descriptor on the remote node. Failure to do so may result in the message being dropped or a reliable connection broken. In the RDMA model, the initiator specifies both the virtual address of the local user buffer and that of the remote user buffer. In this model, a descriptor does not have to be posted on the receiver side corresponding to every message. VIA provides the RDMA Write and RDMA Read features. In the RDMA Write operation, the node writes directly to the remote node's memory. Similarly in the RDMA Read operation, the node reads directly from the remote node's memory. The RDMA Read is an optional VIA feature. Hence, the work done in this paper exploits only the RDMA Write feature of VIA.

Since the introduction of VIA, many software and hardware implementations of VIA have become available. B-VIA [2], M-VIA [10] and GigaNet VIA [9] are among these implementations. In this paper, we use GigaNet VIA, a hardware implementation of VIA for experimental evaluation.

3 Design Issues for RDMA-based Collective Communication Operations

The idea behind using RDMA for collective communication is to use the illusion of shared memory created by RDMA. The RDMA mechanism and memory registration constraints, imposed by VIA open up several major issues for designing a RDMA-based collective communication library. In this section, we discuss the design issues and present some solutions. In the subsequent sections we discuss the design choices for the particular collective communication operation and its implementation.

3.1 Buffer Registration and Address Exchange

The communication buffers need to be registered in VIA. In our scheme, we have the option to register the buffers either statically before the operation or dynamically during the operation.

Static Buffer Registration: We statically register a contiguous region in memory for each communication group (when the communication group is being created) and type of collective operation. This contiguous region is split into fixed size buffers (blocks). The blocks are used in a consecutive order. Since the memory allocated is contiguous, only the address of the first block needs to be communicated to the other nodes, during the initialization phase. The length of the buffer space is the same for all the nodes in the communication group for a given operation. Since the buffers are pre-registered during initialization, the data has to be copied from these registered buffers to the user buffers when they become available. Hence, there is a copying cost involved.

Dynamic Buffer Registration: In the dynamic registration scheme, we allow the use of non-contiguous buffers. The buffers are registered and the buffer address is exchanged at the start of the collective operation instance. We have the additional overhead of an extra round-trip time for the buffer address communication with the destination set in the collective operation before sending the actual data to the destination set. However, in this scheme, the user buffers are available during the operation and can thus be registered. There is no additional copying cost involved. This scheme is also called the *rendezvous* scheme.

For all our RDMA-based collective operation implementations, we use the static buffer management scheme when the data size is less than *5KB*. For smaller messages less than *5KB*, the memory copy time is less and hence the static scheme proves to be more beneficial than the dynamic scheme (which has an address exchange overhead). For greater data size, the copying time increases drastically and hence using the dynamic scheme proves to be more optimal. MVICH, which is an implementation of MPI on VIA, also uses the dynamic registration scheme for messages greater than *5KB*. For messages lesser than *5KB*, MVICH communicates using the send/receive model.

3.2 Data Validity at the Receiver end

RDMA write is receiver transparent. It does not require that the receiver post a descriptor or perform any action in anticipation of the incoming data and the receiver process receives no indication that any new data has been written. When the destination needs the data it goes to the memory location and fetches the data from there by performing a local read operation. Thus, we need a mechanism for indicating to the receiver that the data in the memory is valid data. This can be done in multiple ways. We briefly discuss some of these methods in [7]. For our implementations, we indicate data validity by a byte, having a special value, which is attached to the end of the data when it is sent to the destination node. The destination knows when and how much data is arriving and thus it checks the value of the byte at the end of data and determines the validity of the data.

3.3 Safely reusing the buffers

In the static buffer allocation scheme, we register a fixed number of buffers which may be reused when needed. The buffers are contiguous in nature and are used contiguously. A receiver belonging to a specific communication group performing a particular collective operation, knows exactly which buffer is going to be reused by the sender. Before reusing the first buffer, the sender waits for a notification from the receiver. The receiver sends the notification by RDMA writing a known value at a special location in the notification buffer.

4 RDMA-based Broadcast

In this section, we briefly discuss the design choices with relation to the RDMA-based broadcast. For detailed discussion of all the following subsections, refer to [6].

4.1 The Broadcast Algorithm

The broadcast collective operation distributes the data present at one node (called as the *root*) to all the other nodes in the communicator. Higher level libraries use various algorithms to implement the broadcast operation. Libraries like MVICH use the binomial algorithm which is very efficient for small-large clusters. Some libraries use the linear method for implementing broadcast in very small clusters. Currently, binomial broadcast is implemented using the send/receive primitives. We implement the same algorithm in RDMA and we compare the performance of the two. The binomial broadcast follows a recursive method, wherein the destination node on receiving the data, becomes one of the source nodes and forwards the data to other nodes. Figure 1 shows binomial broadcast in an 8 node cluster. $logN$ steps are needed for performing the binomial broadcast, where N is the number of nodes. The data structures and the working of the algorithm is discussed in the following subsections.

4.1.1 Registration of buffers : Message Size $<$ *5KB*

We divide a contiguous registered memory (called *broadcast buffer*) into fixed blocks of size *block_size*. The blocks

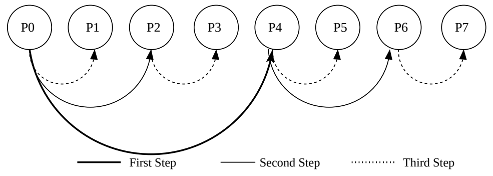


Figure 1. Broadcast using Binomial Algorithm for 8 node cluster

are numbered from 0 onward. In addition to the broadcast buffer, every node also reserves a buffer called notification buffer which is used to indicate the safe reuse of the broadcast buffers. The size of the notification buffer in bytes is equal to the number of nodes in the communication group involved in the broadcast operation. The address of the broadcast and notify buffers are exchanged during the initialization time. The buffers are initialized to -1 during the initialization time.

4.1.2 Registration of buffers : Message Size $>$ *5KB*

Since we adopt the *rendezvous* dynamic buffer registration approach for messages larger than *5KB*, which is similar to the one adopted by MVICH, we discuss the remainder of the section with respect to the static broadcast scheme for messages lesser than *5KB*. We choose *5KB* as the boundary for the static registration scheme to facilitate fair comparison of our RDMA-based collective communication with the MVICH send/receive-based collective communication.

4.1.3 Data Validity at the Receiver end

To understand the working of the RDMA-based broadcast, consider Figure 2 with 4 nodes P0, P1, P2, P3, where node P0 is the root and the broadcast instance shown is between the nodes P0 and P2.

For every communication group, we have a static counter called *broadcast counter* which is incremented by 1 for every broadcast operation, by every node within that communication group. Consider the first broadcast of data size *block_size/2* bytes. The *broadcast counter* for the first broadcast is set to 1. The sender appends the *broadcast counter* byte at the end of the data to be written. The root P0 can RDMA write the data of size *block_size/2 + 1*, which includes the *broadcast counter*, to block 0 of node P2. For a communication group, every node is involved in the collective operation. So, every node can keep track of the number of blocks used for that particular collective operation.

The data is written in a *bottom-fill* manner. To write 8 bytes in a 10-byte block in a *bottom-fill* manner, we start writing these 8 bytes from the 2nd byte onward in the block. Hence the data is always filled in the lower portion of the block.

Since the receiver shares the communication group with the sender, the *broadcast counter* at the sender and receiver will have the same value. Thus, the receiver can poll for the *broadcast counter* on the last byte of the received block and check for the validity of the data.

If the data to be sent is greater than the *block_size*, the data is split up into blocks of size *block_size - 1* with the *broadcast counter* appended to each block of data. Figure

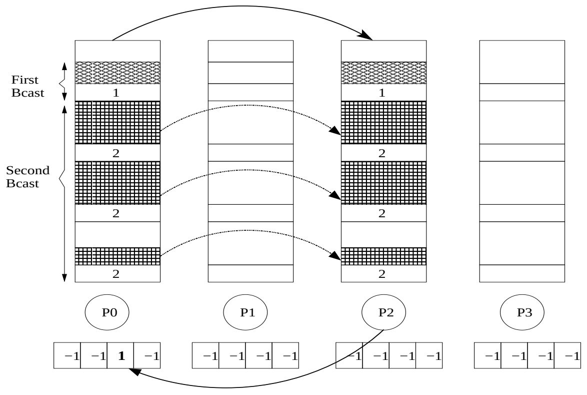


Figure 2. Two Consecutive RDMA-based Broadcast instances followed by a notification

2 also shows the second broadcast of *2 * block_size* bytes, which divides the data into three blocks of size *block_size - 1*, *block_size - 1* and *2* bytes respectively, with the *broadcast counter* of 2 attached for data validity. An intermediate node forwards the message only after it receives all the blocks of that message.

Writing large messages by breaking them into blocks at the lowest level enables pipelining of messages and overlapping of the copy to the user buffers at the destination. However, there is a trade-off involved between the *block_size* and number of RDMA writes. It takes 1 RDMA write to send each block. If the *block_size* is too large, the number of RDMA writes will be low but the copying cost to the buffers will be high. If the *block_size* is small, the copying cost will be low, but the number of RDMA writes and contention at the switches and NICs will be high. The processing of a large number of *RDMA writes* might offset the benefit obtained by overlapping the copies. Thus, it is desirable to find an optimal *block_size* where the cost of processing multiple *RDMA writes* does not kill the benefit achieved by overlapping memory copies. In our results section, we evaluate our RDMA-based broadcast algorithm with different *block_sizes* to obtain an optimal one.

4.1.4 Buffer reusing

To bring about safe reuse of the broadcast buffer blocks, we use the pre-registered notification buffers. The sender, before sending the data, waits for the receiver to write a special value into its notification buffer. Figure 2 shows the notification from node P2 to node P0, before the third broadcast, which requires the reuse of the first buffer. Before writing to the notification buffer, the receiver resets the last byte (*broadcast counter*) of each block in its broadcast buffer, to clear the *broadcast counter* values of the previous broadcast operation.

Writing data in a *bottom-fill* manner requires us to reset only the last byte of each block. If data was written starting from the top of the block, the *broadcast counter* would have been at an arbitrary location depending on the size of data being sent and hence all the bytes of all blocks would have to be reinitialized which may be an expensive operation.

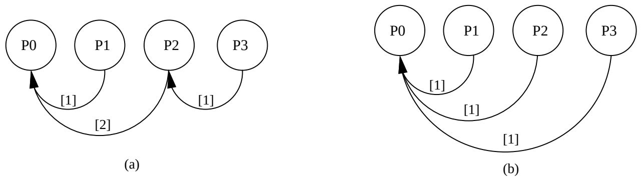


Figure 3. RDMA-based allreduce: (a) Degree-1, (b) Degree-3

4.2 RDMA-based AllReduce

AllReduce is a global reduction operation which combines values from different nodes based on the reduction operation and communicates the result back to all the nodes involved in the operation. It is generally implemented as a combination of the reduce operation (where the result is present at the *root* node) and the broadcast operation. The reduce algorithm can be implemented in different ways. Current libraries like MVICH implement reduce using a recursive binomial algorithm. We introduce a new algorithm for reduce called the *Degree-k* algorithm. The new algorithm when implemented for the allreduce operation with the RDMA mechanism was found to give good performance.

Definition: An *Degree-k tree-based Reduce* defines a tree where any node can receive messages from at most k nodes in any step of the reduce operation. The variable k is a *(power of 2) - 1* value. For a cluster of size N, where N is a *power of 2*, we can use all those *Degree-k tree-based* reduce schemes, where k is *(power of 2) - 1* and $k < N$. The binomial reduce algorithm is similar to the Degree-1 tree-based reduce algorithm. Since, most computing clusters generally have a *power of 2* size, we constrain the Degree-k tree-based reduce scheme to a cluster having *power of 2* nodes.

We implement a Degree-k RDMA-based allreduce as a combination of the Degree-k reduce and the binomial broadcast with the RDMA mechanism. To understand the Degree-k RDMA-based allreduce concept, let us consider a 4 node cluster. An allreduce operation on such a cluster can be implemented using Degree-1 or Degree-3 RDMA-based allreduce scheme. Figure 3 shows the tree for 4 nodes P0, P1, P2, and P3 having *ids 0, 1, 2 and 3* respectively. The square brackets indicate the *step number* for that node. In a Degree-1 RDMA-based allreduce scheme, every node will receive data from at-most 1 node in each step. A receiver node chooses the order of evaluating the data based on the ascending order of the ranks of the sending nodes. Hence, in the first step P1 and P3 send data to P0 and P2 respectively, which perform the computation. The computed result is forwarded by P2 to P0. The final result, calculated at P0, is then broadcast to all the other nodes involved in the operation.

In a Degree-3 RDMA-based allreduce (Figure 3b), P0 waits for P1, P2 and P3 to send data and then computes the result. P0 chooses the order of evaluating the data based on the ascending order of the ranks of the sending nodes.

A Degree-3 RDMA-based allreduce for a 32 node cluster has the tree as described in Figure 4. There is a trade-off involved between the number of steps in the Degree-k

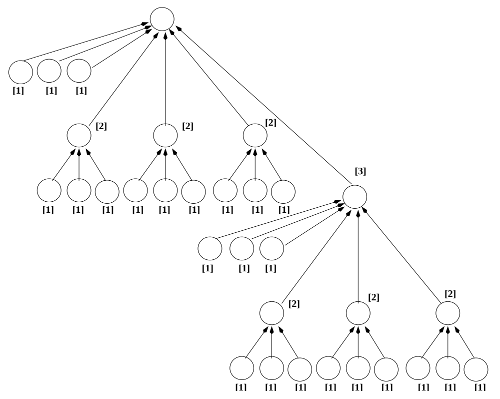


Figure 4. Demonstration of steps in a Degree-3 RDMA-based allreduce for 32 nodes

RDMA-based allreduce collective operation and the overhead incurred by the node in performing the reduction operation. For example, in a Degree-1 RDMA-based allreduce scheme for a 4 node cluster, there are 2 steps and in each step, a receiver node receives only 1 message and hence performs only 1 operation. In a Degree-3 RDMA-based allreduce scheme for a 4 node cluster, there is 1 step but the receiver nodes does 3 operations. So, depending upon the number of nodes, number of steps and the number of operations involved, we can choose different Degree-k RDMA-based allreduce algorithms.

The design solutions for all the Degree-k RDMA-based allreduce algorithm are the same. We explain the design solutions for a Degree-1 RDMA-based allreduce algorithm in the following few subsections.

4.2.1 Registration of buffers: Message Size $<$ *5KB*

We allocate a contiguous registered section of memory called *allreduce buffers*, split into *block_size* of *(5K + 1)* bytes, because *5KB* is the maximum size of data that can be transferred in the static scheme. Also, the total memory region reserved need not be greater than $block\_size * N$, where N is the number of processes in the communicator. This is because, in the Degree-(*N-1*) RDMA-based allreduce algorithm, a maximum of $N - 1$ processes can write to a receiver node. Figure 5a shows the 4 nodes P0, P1, P2 and P3 with *ids 0, 1, 2, 3* respectively, each having 4 contiguous blocks of memory reserved and registered for the allreduce operation.

4.2.2 Registration of buffers: Message Size $>$ *5KB*

For messages greater than *5KB*, we follow the *rendezvous* scheme (described in the broadcast section). The remainder of the allreduce discussion will be for messages less than *5KB*.

4.2.3 Data Validity at the Receiver end

Consider the node P1 sending data to node P0 and node P3 sending data to node P2. The node P1, with *id 1*, RDMA writes the data to block #1 of the receiver's allreduce buffer. A node always writes the data to the block having the same number as its *id*. Any node with any *id* can write to any other node's allreduce buffer at a location indicated by its *id*. This indicates to the receiver the identification of the sender of the data and also enables an ordered evaluation of the data.

Node P1 sends the entire data in 1 single RDMA write to the node P0. Similar to the broadcast operation, the allreduce operation has a static *allreduce counter* which is appended to the block of data when it is sent. The data is written in a *bottom-fill* manner. The receiver performs the required operation and the result is stored in the same location as that of the latest received data from the node having the greatest id. Figure 5a shows node P1 with *id 1* and node P3, with *id 3*, writing the data to node P0's block #1 and to node P2's block #3 respectively. Assuming this is the first allreduce, the *allreduce counter* is set to 1.

Figure 5b shows node P0 and node P2 performing the operation and writing the intermediate results in blocks #1 and block #3 respectively. Data is not broken down into smaller blocks and sent. The overhead of copying, assembling and packing data has been found to be larger than the overhead of sending the entire data in a single RDMA write operation [5, 6].

In the second step of the Degree-1 RDMA-based allreduce, node P2 with *id 2* RDMA writes its latest computed result to block #2 of node P0's allreduce buffer, with the *allreduce counter* of 1 attached at the end of the data. P0 performs the operation on the newly received data from node P2 and its own computed result obtained in the previous step. The result of this operation is stored in block #2 at node P0's allreduce buffer. The result is copied by node P0 to its user receive buffer after its done with its final computation. The result is broadcast from this user buffer to all the nodes.

4.2.4 Buffer reusing

There is no extra step needed to indicate safe buffer reusability because in an allreduce operation for a given communicator, the second allreduce operation starts only after all the nodes have received the broadcast results of the first operation.

5 Analytical Models

In this section we present some important parameters of the analytical model for RDMA-based binomial broadcast and the Degree-k RDMA-based allreduce. The analytical models are described very briefly due to space constraints. For detailed understanding of the analytical models, refer to [6, 5]. These models help us estimate the performance of collective operations for large scale systems.

5.1 Binomial RDMA-based broadcast Analytical model

A message transfer in a binomial RDMA-based broadcast operation consists of many events and overheads in the communication stack. At the sender side, there is a copying overhead due to the need to copy data into registered buffers (*Tc*), the MPI library overhead (*Tm*), the cost involved in posting the send descriptor (*Td*), DMAing the data (*Ts*) from the host, the processing by the NIC (*Tn*) and the time for the

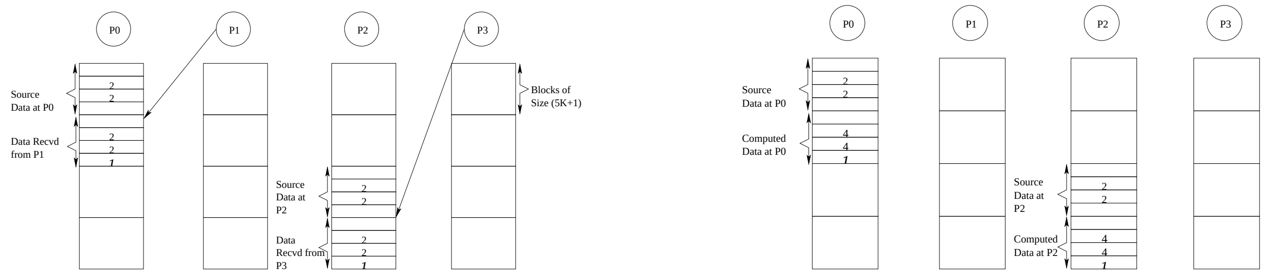


Figure 5. (a) Step 1 of Degree-1 RDMA-based allreduce, (b) Reduce Computation at P1 and P2

acknowledgment message (*Ta*). The data is then transmitted (*Tt*) to the destination. If the destination is connected to the source via a switch, the messages are sequentialized at that switch. Data can be broken into smaller blocks and sent which enables pipelining and overlaps the memory copying with the sending side events. When the receiver NIC receives the data, it processes it and obtains the destination memory address (in case of RDMA write), to which the data is then DMAed. Copying into the user buffer takes place after the destination has forwarded the data to the other nodes if necessary. *Tt* and *Tc* depend upon the total bytes that are being communicated. We assume that *Tn* is less than *Ts*, which is true for most current generations systems. The analytical model for RDMA-based broadcast gives timing estimates for broadcast taking place with varying *block_sizes*. The total broadcast time where the message is broken into *num* blocks, can be represented by the equation: *Tm + Td + 2*(Ts + Tn + Tc) + num*(Tt + Ta)*. For detailed explanation, refer to [6].

5.2 Analytical model for RDMA-based allreduce

For a given set of *power of 2* nodes, we have different *degree-k* algorithms available. Using the analytical model, we can determine an optimal degree k, for a given number of nodes and data size on the basis of the parameters discussed in the previous section. In addition to these parameters, we also have to take into account the time spent in performing the reduction operation on the given data size. This total time can be indicated by *To*. The assumptions made in the previous section hold true for allreduce too. Typically, for large messages *To* is much lesser as compared to *Tt*. However this is not true for small messages having small counts.

A message transfer in an allreduce operation consists of similar events as described in the previous section. The only difference being that, once the data is DMAed by the destination NIC, the reduction operation is performed. The result can then be broadcast to all the nodes in the communication group. The analytical model has various cases based on the values of the above parameters. A detailed analysis of the analytical model along with the timings diagrams and pseudo-code can be found in [6, 5].

For the degree-k RDMA-based allreduce case, the receiver performs reduction operation based on the order of *ids*, starting with the data sent by the node with the lowest *id*. When many nodes are performing an RDMA write to a single destination simultaneously, there might be contention at the NIC. Thus, the order in which the messages arrive at the destination NIC and hence in the destination buffer is indeterminate. Thus, if a NIC is waiting for allreduce data to arrive from the lowest *id* node, there is a fair probability that the data might not arrive first. Hence, the analytical model for RDMA allreduce gives the best and the worst time estimates. The best time estimate assumes that the required data is the first to arrive and assumes the ideal network condition. The worst time estimate assumes that the required data is the last to arrive due to NIC and switch contention. It also assumes that the NIC processing and the DMA startup can't be overlapped due to sharing of the system bus. The best case equation for very large messages is given as: *2*Tc + (Total Number of Steps)*[2*(Ts + Tn) + Tm + Td + Tt*(No. of Sending Nodes in that step) + To]*. For detailed explanation and the other cases such as the analytical model for small messages, etc., refer to [6].

6 Performance Evaluation

In this section, we discuss the results that have been obtained for RDMA-based broadcast and RDMA-based allreduce. We evaluated our implementations on a cluster of 16 nodes, each with a 33MHz PCI bus, 1.0GHz Pentium III machines, 512MB of Main memory and Linux version 2.2.17. The machines are connected using a GigaNet 5300 switch. In addition to the experimental results, we also present results for larger systems using the analytical model.

6.1 Broadcast Performance

The RDMA-based binomial broadcast is compared with the send/receive-based binomial broadcast, provided by the MVICH-1.0 implementation of MPI. The broadcast latency, averaged over 5000 iterations, is calculated between the root node and the last leaf node receiving the message. The last leaf is typically chosen to be the one having the maximum depth from the root in the binomial tree structure.

6.1.1 RDMA-based broadcast V/s Send/Receive-based broadcast on a 16 node cluster

Using the RDMA scheme, a large message is broken into multiple blocks depending upon the *block_size*. MVICH-1.0, in the send/receive-based binomial broadcast, sends data in blocks of *1KB*. To ensure fair comparison, we tested the RDMA-based binomial broadcast with different block_sizes starting with 1025 (1 extra byte for the counter) bytes.

Figure 6 shows a comparison of RDMA-based broadcast with *block_sizes* of 1025, 2049, 3073 and 4097 bytes and the send/receive-based broadcast for small messages for a cluster of 16 node. Small messages from 4-1024 bytes show the same timings as they use a single 1025 byte block to write to the remote node. The difference can be seen in Figure 7. We see that to transmit 4096 bytes with *block_size* of 1025 bytes, we need 4 blocks. For higher *block_sizes*, the number of RDMA writes decrease and so do the timings. A *block_size* of 3073 bytes gives the most optimal result for all message sizes from 1025 to 5000 bytes. RDMA-based binomial broadcast for all the given *block_sizes* gives better performance as compared to the send/receive-based binomial broadcast.

With RDMA-based binomial broadcast with *block_size* of 3073 bytes, we get a benefit of around 19% for smaller message of 4 bytes. For larger messages we see a benefit of around 14% for 4608 bytes.

We also use the broadcast analytical model, presented in the previous section, to estimate the timings for RDMA-based binomial broadcast with varying block sizes. Figure 8 shows the comparison of the analytically obtained RDMA-based broadcast with the experimentally obtained RDMA-based broadcast using *block_size* of 3073 bytes for a 16 node cluster. For all data sizes (4-4096 bytes), the analytically obtained results closely match the experimental ones with an error rate of below 10%.

6.1.2 RDMA-based broadcast V/s Send/Receive-based broadcast on large clusters

We use this analytical mode to estimate the performance of RDMA-based binomial broadcast on 512 and 1024 node systems. We compare the estimated RDMA-based broadcast timings with send/receive-based broadcast for the same cluster size. Extrapolating the timings obtained for broadcast between a pair of nodes, we obtain the latency for the send/receive-based broadcast for large clusters of size N to be approximately $log(N) * t1$, where $t1$ is the time for send/receive-based broadcast between 2 nodes.

Figures 9 and 10 show the comparison graphs for RDMA-based broadcast and send/receive-based broadcast for 512 and 1024 nodes. The RDMA-based broadcast timings have been taken with an optimal block size of 3073 bytes. For 512 nodes, we achieve a benefit of about 21% for *4KB* messages and for smaller messages of 4 bytes, we get a performance benefit of 16%. For 1024 nodes, for *4KB* message size, we again obtain a performance benefit of about 21% and for smaller messages a benefit of 16%.

6.2 RDMA-based AllReduce Performance

The timings for allreduce are an average of 5000 iterations. The operation used was MPI_SUM with the data type MPI_INT of size 4 bytes and the count of the elements was varied from 1 (4 bytes) to 1024 (4096 bytes).

6.2.1 RDMA-based allreduce V/s Send/Receive allreduce on a 16 node cluster

In this paper, we introduced the analytical model for *degree-k* RDMA-based allreduce. Depending upon the network topology, the analytical model gives the best and the worst time estimates for the allreduce operation. We have evaluated the *degree-k* RDMA-based allreduce analytical model for 4, 8, 16 nodes with 4-4608 byte data sizes. The following values have been chosen for the system dependent parameters of RDMA-based allreduce. $Tt\_per\_byte = 0.010\mu s$, $Ts = 2\mu s$, $Tc\_per\_byte = 0.0027\mu s$, $Tn = 1.52\mu s$, $Td = 0.6\mu s$, $Tm = 1.3\mu s$. Tt and To are calculated depending on the total number of *bytes* and *count* of the operation.

The following results for a cluster of 16 nodes can be referred in [6]. For 16 nodes, we can use the *degree-1, degree-3, degree-7, degree-15* RDMA-based allreduce schemes. We have obtained the best and worst case timings for all these cases for message sizes varying from 4-4608 bytes. Further, we have compared and proved that the experimental results obtained for all the above *degree-k* RDMA-based allreduce schemes, lie between the best and the worst case timings predicted by our analytical model. The largest deviation from the actual timings obtained is around 8% for 512 bytes in the *degree-15* analytical model for a 16 node cluster. For small data sizes like 4 bytes, the analytical model gives the *degree-3* RDMA-based allreduce scheme as the optimal scheme. For larger messages, the analytical model shows *degree-1* RDMA-based allreduce to be the most optimal one. This is because in the *degree-3* RDMA-based allreduce case, 3 nodes write to 1 node and 3 operations are done at the receiving node. As the data size increases, the number of operations increase and computation becomes very expensive. The *degree-1* RDMA-based allreduce fares better as the computation is distributed to a greater number of the nodes. We verified both sets of results practically. [6] contains comparison graphs showing the experimental results obtained by all the above *degree-k* RDMA-based algorithms for a 16 node cluster. We summarize the results in Figure 11. For a 4 node cluster, the *degree-3* RDMA-based allreduce scheme performs well for smaller messages till 1024 bytes. For an 8 node cluster, we obtain a slightly improved performance if we use *degree-7* RDMA-based allreduce for very small messages. For messages above 1024 bytes, *degree-1* RDMA-based allreduce always gives the best performance.

Figure 12 compares the results of send/receive-based binomial allreduce, most optimal *degree-k* RDMA-based allreduce and *degree-1* RDMA-based allreduce. The most optimal *degree-k* RDMA-based allreduce uses *degree-3* algorithm for small messages and *degree-1* algorithm for messages greater than *1KB*. The *degree-1* allreduce is exactly the same as the binomial allreduce algorithm. We see that the RDMA-based binomial algorithm (i.e *degree-1*) always performs better than the send/receive-based binomial algorithm. The *degree-3* RDMA-based algorithm outperforms both the send/receive-based and RDMA-based binomial allreduce algorithms for small messages (4-1024 bytes). On a 16 node cluster, we obtain a 38% benefit for 4 byte messages, when we use the *degree-3* RDMA-based allreduce. For larger messages of size *4KB*, we get a 9%

improvement on using the optimal *degree-1* RDMA-based allreduce scheme. The benefits obtained are due to an optimal algorithm implemented with an efficient RDMA mechanism.

6.2.2 RDMA-based allreduce V/s Send/Receive-based allreduce for large clusters

We evaluated the analytical model for clusters of 512 and 1024 nodes. An analysis of the best and worst case timings shows that *degree-3* RDMA-based allreduce performs optimally for small messages (4-1024 bytes) and *degree-1* RDMA-based allreduce performs the best for larger messages (1025-5000 bytes) (Figure 11). Thus we can generalize by saying that a *degree-3* RDMA-based allreduce algorithm should give good performance for smaller data size (from 4 to 1024 bytes) and a *degree-1* RDMA-based allreduce scheme can be used for larger data sizes while implementing the reduce part of the allreduce collective operation.

We also use the RDMA-based allreduce analytical model to predict the performance achievable in large clusters. We obtain the send/receive-based binomial allreduce latency by extrapolating the allreduce latency between 2 nodes. If the time for allreduce between 2 nodes is *t1*, then the time taken for the same allreduce between N nodes, which involves $log(N)$ steps is approximately *log(N) * t1*.

Figure 13 shows the comparison between the estimated send/receive-based binomial, the best and the worst case of the most optimal *degree-k* RDMA-based allreduce and the *degree-1* (binomial) RDMA-based allreduce. We use the most optimal *degree-3* algorithm for message sizes lesser than *1KB* and *degree-1* algorithm for messages greater than *1KB* in the *degree-k* RDMA-based allreduce. The analytical models predicts a 14% performance benefit of the best case of the most optimal *degree-k* RDMA-based allreduce for message size of *4KB* and around 40% for small messages of 4 bytes. When send/receive-based binomial allreduce is compared with the worst case of the most optimal *degree-k* RDMA-based allreduce, we still obtain a benefit of about 35% for 4 byte messages and 14% for *4KB* messages. The *degree-3* RDMA-based allreduce outperforms the RDMA-based binomial as well the send/receive-based binomial algorithm for small messages. Similar timings are obtained for 1024 nodes (Figure 14). The analytical model predicts a benefit of 41% for 4 bytes and 14% for *4KB*.

7 Conclusions and Future work

Traditionally, collective operations have been implemented on the send/receive message passing primitives. In this paper, we introduce a novel method to implement fast broadcast and allreduce communication operations on VIA based clusters using RDMA operations. We implemented *RDMA-based broadcast* using the binomial algorithm which gives a 14% benefit for a 16 node cluster for 4608 bytes and 19% for 4 byte messages as compared to the send/receive-binomial broadcast algorithm. For the *allreduce* operation, we introduced a new concept of *degree-k tree-based* allreduce algorithms which when combined with the RDMA mechanism gives improved performance as compared to the send/receive-based algorithms. A comparison of the most optimal degree-k RDMA-based allreduce with the send/receive-based binomial allreduce gives us a benefit of about 38% benefit for a small data size of 4 bytes and about 9% for data size of *4KB* for a 16 node cluster. We also presented analytical models for broadcast and allreduce that give an estimate of the broadcast time for large clusters and the most optimal *degree-k RDMA-based* allreduce algorithm that can be used for a given cluster and data size. We have used the analytical model to predict the performance benefits achievable by our RDMA implementation of allreduce on large clusters. The predicted performance shows a benefit of about 35-40% for small messages of 4 bytes and around 14% for messages of *4KB* for 512 and 1024 node clusters.

In future, we plan to perform in-depth analysis of the global buffer management for other collective operations. We plan to explore and develop efficient algorithms dealing with user-defined communicators used in context with these collective operations. We also plan to extend this framework to the emerging InfiniBand architecture.

8 Acknowledgments

The work at Pacific Northwest National Laboratory was performed under the auspices of the U.S. Department of Energy (DOE). Pacific Northwest National Laboratory is operated for DOE by Battelle Memorial Institute. This work was supported by the Center for Programming Models (www.pmodels.org) for Scalable Parallel Computing sponsored by the Mathematical, Information, and Computational Science Division of DOE's Office of Computational and Technology Research.

References

[1] Infiniband Trade Association. http://www.infinibandta.org.

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

[3] R. Geijn, D. Payne, L. Shuler, and J. Watts. *A Streetguide to Collective Communication and its Application*. Jan 1996.

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

[5] Rinku Gupta. M.S. Thesis: Efficient Collective Communication using Remote Memory Operations on VIA-Based Clusters, The Ohio State Univeristy, August 2002.

[6] Rinku Gupta, Pavan Balaji, Dhabaleswar K. Panda, and Jarek Nieplocha. Efficient Collective Communication using Remote Memory Operations on VIA-Based Clusters. Technical Report OSU-CISRC-1/03-TR03, The Ohio State University, December 2002.

[7] Rinku Gupta, Vinod Tipparaju, Jarek Nieplocha, and Dhabaleswar K. Panda. Efficient Barrier using Remote Memory Operations on VIA-Based Clusters. In *IEEE Cluster Computing*, 2002.

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

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

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

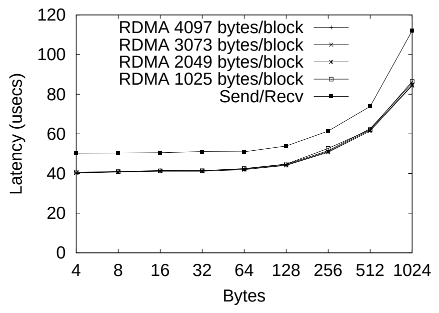


Figure 6. RDMA-based V/s Send/Receive-based broadcast comparison (4-1024 bytes) - 16 Node Cluster

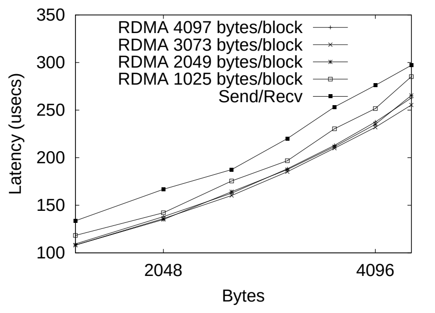


Figure 7. RDMA-based V/s Send/Receive-based broadcast comparison (1025-4608 bytes) - 16 Node Cluster

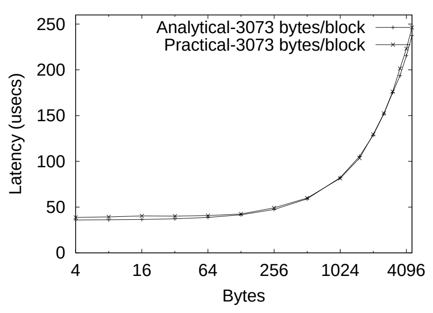


Figure 8. RDMA-based binomial broadcast Analytical V/s Experimental comparison (4-4608 bytes) - 16 node Cluster

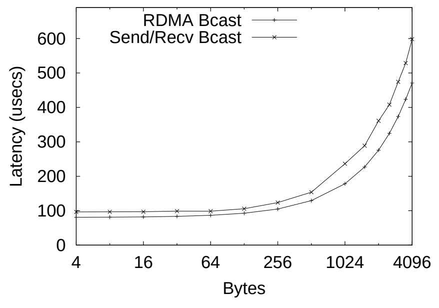


Figure 9. RDMA-based V/s Send/Receive-based broadcast Estimations - 512 node Cluster

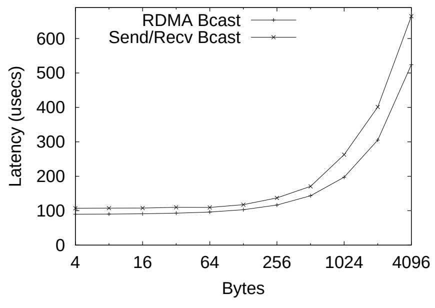
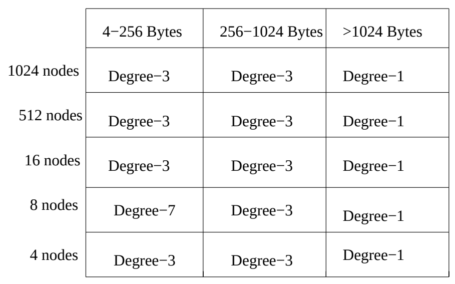


Figure 10. RDMA-based V/s Send/Receive-based broadcast Estimations - 1024 node Cluster



	4−256 Bytes	256−1024 Bytes	>1024 Bytes
1024 nodes	Degree−3	Degree−3	Degree−1
512 nodes	Degree−3	Degree−3	Degree−1
16 nodes	Degree−3	Degree−3	Degree−1
8 nodes	Degree−7	Degree−3	Degree−1
4 nodes	Degree−3	Degree−3	Degree−1

Figure 11. Choosing the Right algorithm for varying configurations

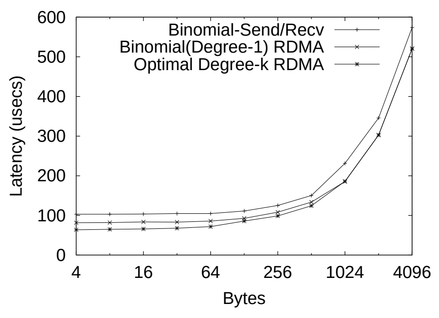


Figure 12. Send/Receive-based binomial, RDMA-based binomial and the most optimal degree-k RDMA-based allreduce comparison - 16 node Cluster

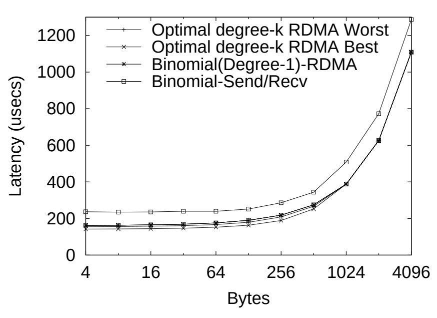


Figure 13. Send/Receive-based binomial, RDMA-based binomial and the most optimal degree-k RDMA-based allreduce Estimations - 512 node Cluster

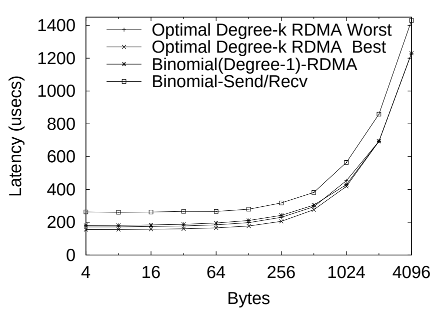


Figure 14. Send/Receive-based binomial, RDMA-based binomial and the most optimal degree-k RDMA-based allreduce Estimations - 1024 node Cluster