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

SAYANTAN SUR, UDAY KUMAR REDDY BONDHUGULA, AMITH MAMIDALA,
HYUN-WOOK JIN AND D. K. PANDA

Technical Report
OSU-CISRC-5/05-TR32

High Performance RDMA Based All-to-all Broadcast for InfiniBand Clusters *

Sayantan Sur, Uday Kumar Reddy Bondhugula, Amith Mamidala, Hyun-Wook Jin,
and
Dhabaleswar K. Panda

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

Abstract. The All-to-all broadcast collective operation is commonly used in parallel scientific applications. This collective operation is called `MPI_Allgather` in the context of MPI. Contemporary MPI implementations use the Recursive Doubling and Ring algorithms for implementing this collective on top of MPI point-to-point calls. This leads to several performance bottlenecks. Depending on message size and number of processes involved, the overheads include extra message copies, protocol handshake and multiple buffer registration costs. In this paper, we propose a design of All-to-All broadcast using the Remote Direct Memory Access (RDMA) feature offered by InfiniBand, an emerging high performance interconnect. Our design eliminates the overhead of protocol handshake and multiple buffer registrations. Further, our design aims at cutting down the copy cost by dynamically choosing an optimal threshold from a copy-based approach to a zero-copy one as the collective progresses. Our results indicate that latency of the All-to-all Broadcast operation can be reduced by 30% for 32 processes and a message size of 32 KB. In addition, our design can improve the latency by a factor of 4.75 under no buffer reuse conditions for the same process count and message size.

1 Introduction

Cluster based computing systems are becoming popular for a wide range of scientific applications, owing to their superior cost-performance ratio. These systems are typically built from commodity PCs connected with high speed Local Area Networks (LANs) or High Performance Interconnects [4]. MPI [1] has become the *de-facto* standard in writing parallel scientific applications which run on these clusters. MPI provides both *point-to-point* and *collective* communication semantics. Many scientific applications use collective communication primitives to synchronize or exchange data with multiple processes [2, 11]. Of these collective operations, All-to-all broadcast (`MPI_Allgather`) is an important one used in many applications such as matrix multiplication, lower and

* This research is supported in part by Department of Energy's grant #DE-FC02-01ER25506, National Science Foundation's grants #CCR-0204429 and #CCR-0311542; grants from Intel and Mellanox; and equipment donations from Intel, Mellanox, AMD, Apple and Sun Microsystems.

upper triangle factorization, solving differential equations, and basic linear algebra operations.

InfiniBand [7] is emerging as a high performance interconnect for interprocess communication and I/O. It provides powerful features such as Remote DMA (RDMA) which enables a process to directly access memory on a remote node. Traditional mechanisms of implementing All-to-all broadcast use point-to-point communication primitives [19]. These designs are not optimal. Depending on the message size, they incur the overhead of message copying and protocol handshake. In this paper, we propose efficient design of All-to-all broadcast using RDMA capability of InfiniBand. The basic idea in our design is to bypass the multiple layers in the software stack and overlap the protocol handshake with the communication.

We have implemented and incorporated our designs into MVAPICH [15], a popular implementation of MPI over InfiniBand used by more than 210 organizations world wide. MVAPICH is an implementation of the Abstract Device Interface (ADI) for MPICH [5]. MVAPICH is based on MVICH [10]. Our design directly uses RDMA for collective operations. We cut down on several message copies and avoid protocol handshake. In addition, our design reduces unnecessary cost of multiple buffer registrations. Our performance evaluation reveals that our designs improve the latency of `MPI_Allgather` on 32 processes by 30% for 32 KB message size. Additionally, our RDMA design can improve the performance of `MPI_Allgather` by a factor of 4.75 on 32 processes for 32 KB message size, under no buffer reuse conditions.

The rest of this paper is organized as follows: in section 2, we provide an overview of the InfiniBand Architecture, `MPI_Allgather` and existing algorithms for it. We describe in detail our motivation in section 3. In section 4, we describe our RDMA based design in detail. We follow up the design discussion with experimental evaluation in section 5. We describe related work in section 6. Finally, this paper concludes in section 7.

2 Background

2.1 Overview of InfiniBand Architecture

The InfiniBand Architecture [6] defines a switched network fabric for interconnecting processing and I/O nodes. It provides a communication and management infrastructure for inter-processor communication and I/O. In an InfiniBand network, hosts are connected to the fabric by Host Channel Adapters (HCAs). InfiniBand utilities and features are exposed to applications running on these hosts through a *Verbs* layer. By calling these verbs layer functions, applications can access the InfiniBand network.

InfiniBand uses a queue-based model. A Queue Pair in InfiniBand consists of two queues: a send queue and a receive queue. The send queue holds instructions to transmit data and the receive queue holds instructions that describe where received data is to be placed. Communication operations are described in the Work Queue Requests (WQR), or descriptors, and submitted to the work queue. The completion of WQRs is reported through Completion Queues (CQs). Once a work queue element is finished, a completion entry is placed in the associated completion queue. Applications can check the completion queue to see if any work queue request has been finished. InfiniBand supports different classes of transport services. In current products, Reliable Connection (RC) service and Unreliable Datagram (UD) services are supported.

InfiniBand Architecture supports both channel semantics and memory semantics. In channel semantics, send/receive operations are used for communication. In memory semantics, InfiniBand provides Remote Direct Memory Access (RDMA) operations, including RDMA Write and RDMA Read. RDMA operations are one-sided and do not incur software overhead at the remote side. Write Gather and Read Scatter are supported in RDMA operations. RDMA Write operation can gather multiple data segments together and write all data into a contiguous buffer at the receiver end. Gather/scatter features are very useful to transfer noncontiguous data. The Gather/Scatter feature not only reduces the startup costs, but also increases network utilization. RDMA Write with Immediate data is also supported. With Immediate data, a RDMA Write operation consumes a receive descriptor and then can generate a completion entry to notify the remote node of the completion of the RDMA Write operation. Regardless of channel or memory semantics, InfiniBand requires that all communication buffers to be "registered". This buffer registration is done in two stages. In the first stage, the buffer pages are pinned in memory (i.e. marked unswappable). In the second stage, the HCA memory access tables are updated with the physical addresses of the pages of the communication buffer.

2.2 `MPI_Allgather` Overview

`MPI_Allgather` is an All-to-all broadcast collective operation defined by the MPI standard [13]. This operation is used to gather data from every process in a communicator and distribute the collected data to all. `MPI_Allgather` is a blocking operation (i.e. control does not return to the application until the receive buffers are ready with data from all processes). In addition, `MPI_Allgather` can be used only if all participating processes have a fixed length of buffer to send and gather data. After `MPI_Allgather` is over, the block of data sent from jth process is received by every process and is placed in the jth block of the receive buffer.

2.3 Related Algorithms and their Cost Models

Several algorithms can be used to implement `MPI_Allgather`. Depending on system parameters and message size, some algorithms may outperform the others. Currently, MPICH [5] 1.2.6 uses the Recursive Doubling algorithm for power-of-two process numbers and up to medium message sizes. For non-power of two processes, it uses the Bruck's algorithm [3] for small messages. Finally, the Ring algorithm is for large messages [18].

In this section, we provide a brief overview of the Recursive Doubling and Ring algorithms. We will use these algorithms in our RDMA based design.

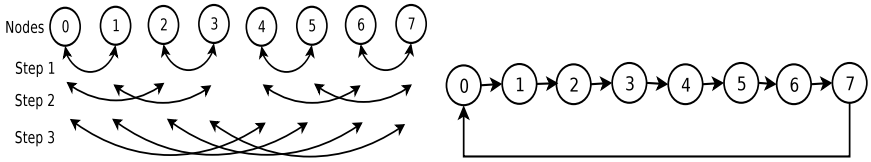
Recursive Doubling In this algorithm, pairs of processes exchange their buffer contents. But in every iteration, the contents collected during all previous iterations are also included in the exchange. Thus, the collected information *recursively doubles*. Naturally, the number of steps needed for this algorithm to complete is $log(p)$, where p is the number of processes. Figure 1(a) depicts the various stages of this algorithm. The communication pattern is very dense, and involves one half of the processes exchanging messages with the other half. On a cluster which does not have constant bisection bandwidth, this pattern will cause contention. The total communication time of this algorithm is:

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

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

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

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



(a) Recursive Doubling Algorithm (b) Ring Algorithm

Fig. 1. MPI_Allgather Algorithms

3 Can RDMA benefit Collective Operations?

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

3.1 Bypass intermediate software layers

Most MPI implementations [5] implement MPI collective operations on top of MPI point-to-point operations. This is shown in Figure 2(a). The MPI point-to-point implementation in turn is based on another layer called the ADI (Abstract Device Interface). This layer provides abstraction and can be ported to several different interconnects. For example, the `p4` device is an implementation over TCP/IP, `shmem` for shared memory communication and so on. As can be observed from the figure, the communication calls pass through several software layers before the actual communication takes place. This can add unnecessary overhead. On the other hand, if collectives are directly implemented on top of the InfiniBand RDMA interface, all these intermediate software layers can be bypassed.

3.2 Reduce number of copies

High-performance MPI implementations, MVAPICH [15], MPICH-GM [14] and MPICH-QsNet [16] often implement an eager protocol for transferring short and medium-sized messages. In this eager protocol, the message to be sent is copied into internal MPI buffers and is directly sent over to an internal MPI buffer of the receiver. This causes two copies for each message transfer. For a collective operation, there are either $2 * log(p)$ or $2 * (p - 1)$ sends and receives (every send has a matching receive). It is clear that as the number of processes in a collective grows, there are increasingly more and more message copies. Thus, collectives based on MPI point-to-point operations have lots of message copies. Instead, with RDMA based design, messages can be directly transferred without undergoing several copies. But in order to utilize RDMA, the source process needs to know the destination memory address. This information can be provided using our design as explained in section 4.

3.3 Reduce Rendezvous handshaking overhead

For transferring large messages, high-performance MPI implementations often implement the Rendezvous Protocol. This protocol is illustrated in Figure 2(b). In this protocol, the sender sends a `RNDZ_START` message. Upon its receipt, the receiver replies with `RNDZ_REPLY`. This reply contains the memory address of the destination buffer. Finally, the sending process sends the `DATA` message directly to the destination memory buffer and issues a `FIN` completion message. By using this protocol, zero-copy message transfer can be achieved.

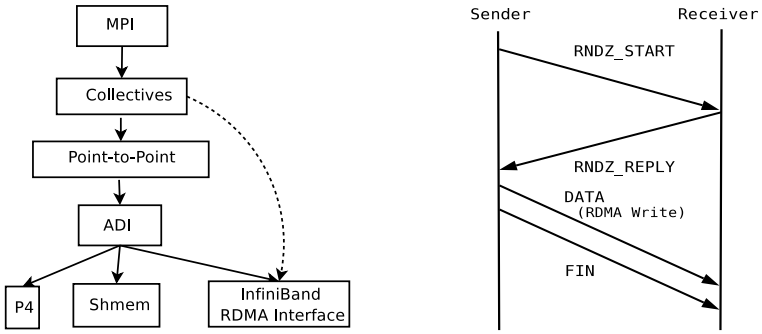
This protocol provides high performance for MPI point-to-point communication, but imposes bottlenecks for MPI collectives based on point-to-point design. The processes participating in the collective need to continuously exchange addresses. However, these address exchanges are redundant. Once the base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

3.4 Reduce Cost of Multiple Registrations

InfiniBand [6], like most other RDMA capable interconnects, requires that all communication buffers be registered with the InfiniBand HCA. This "registration" actually involves locking of pages into physical memory and updating HCA memory access tables. After registration, the application receives a "memory handle" with keys which can be used by a remote process to directly access the memory. Thus, for performing each send or receive, the memory area needs to be registered.

Collective operations implemented on top of point-to-point calls would need to issue several MPI sends or receives to different processes (with different array offsets). This will cause multiple registration calls. For current generation InfiniBand software/hardware stacks, each registration has high setup overhead of around 90 μs (section 5). Thus, point-to-point implementation of collectives requires multiple registration

calls with significant overhead. However, the RDMA based design would need only *one* registration call. The entire buffer passed to the collective call can be registered in one go. Thus, this will eliminate unnecessary registration calls.



(a) Software Layers in MPICH (b) Rendezvous Protocol

Fig. 2. Motivation for RDMA based MPI_Allgather Design

4 Proposed RDMA Based All-to-all Broadcast Design

In this section, we describe our RDMA based design in detail. The proposed implementation path for our RDMA based design is shown as a dashed line in Figure 2(a). By choosing to implement the All-to-all broadcast directly on the InfiniBand RDMA interface, we can avoid the extra software overhead caused by several layers. Since we also bypass the point-to-point implementation, we can avoid the cost of extra message copies. Additionally, by choosing this approach, we have complete knowledge of the collective communication buffer. This information can be used to eliminate multiple registration costs, by just issuing *one* registration call for the entire buffer. Further, the knowledge of the base address allows us to exchange buffer addresses (required for zero copy) only once, and not repeatedly use the Rendezvous protocol (as in the point-to-point based design).

However, before we use RDMA, we have to deal with several design choices that are posed by the RDMA semantics.

4.1 RDMA Design Choices: Copy Based or Zero Copy

As stated in section 2.1, InfiniBand (like other modern RDMA capable interconnects) requires communication buffers to be registered. Thus, for transferring a message, either (1) the message is copied to a pre-registered buffer, or (2) the message buffer itself is registered at both sender and receiver ends. Registration is an expensive operation. In addition, for performing RDMA Write, the sending process needs to know the memory address of the receiving buffer. On current generation InfiniBand hardware, the address exchange phase costs around 10μs. Since the registration operation is expensive, it is not

feasible for smaller message sizes. Additionally, the copy-based approach avoids on-the-fly registration and address exchange costs. On the other hand, the cost for copying large sized buffers can be prohibitively high.

The above trade-offs present two different approaches: Copy based and Zero copy. The former is suitable for small message sizes while the latter is good for larger messages. Now, we discuss the RDMA design for the two algorithms stated in section 2.3.

4.2 RDMA-based Design for Recursive Doubling

We propose a RDMA based design for Recursive Doubling (RD) algorithm. In RD, the size of the message exchanged by pairs of nodes doubles each iteration along with the distance between the nodes as shown in Fig 1(a). If m is the message size contributed by each process, the amount of data exchanged between two processes increases from m in the first iteration to $\frac{mp}{2}$ in the $\log(p)^{th}$ iteration. As we observed in section 4.1, the optimal method to transfer short messages is copy based and for longer messages, we need to use zero copy. However, since in the RD algorithm, the actual message size in each iteration changes, we also have to dynamically switch between copy based and zero copy protocols to achieve an optimal design.

Hence, we switch between the two design alternatives at an iteration k ($1 \leq k \leq \log(p)$) such that the message size being exchanged, $2^{k-1}m$, crosses a fixed threshold M_T. The threshold M_T is determined empirically. Hence, message exchanges in the first k dimensions use a copy-based approach, and those in higher dimensions from $k+1$ through $\log(p)$ use a zero copy approach. It is to be noted that if m itself is greater than M_T, we end up using a zero-copy approach for all the iterations. Similarly, if $\frac{mp}{2} \leq M_T$, the above approach leads to a purely copy-based one. Hence, both approaches are used and a switch occurs between them only when a small message is being gathered over a large number of processes. We find that an M_T of 4 KB is optimal for our experimental system as described in section 5.

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

Collective Buffer: We refer to the buffer used for the copy-based approach as the collective buffer. This is registered at communicator initialization time. Processes exchange addresses of their collective buffers also during that time. Some pre-defined space in the collective buffer is reserved to store the peer addresses and completion flags required for zero-copy data transfers. Data sent in any iteration comprises data received in all previous iterations along with the process' own message. When using the copy-based approach, we need to thus copy the received data from the collective buffer to the user's receive buffer immediately at the end of each iteration. This ensures that whenever we make a switch to the zero-copy approach, all the necessary data required to be sent in the higher iterations is present in the user's receive buffer.

Buffering Scheme: In RD, data is always sent from and received to contiguous locations in either the collective buffer or the user's receive buffer. Hence, RDMA writes are sufficient for implementation. Fig 3 shows the buffering scheme at a node with rank five for RD on eight processes when a pure copy-based approach is used. Since the amount of data written to a collective buffer cannot exceed M_T, the collective buffer never needs to be more than $2M_T$ which is 8 KB (ignoring space for peer addresses

and completion flags) for a single Allgather call. The position of the tail flags in the collective buffer is not constant for different Allgather message sizes it is used for. Hence, the entire portion of the buffer that was used for the previous `MPI_Allgather` call needs to be cleared before it is used for another `MPI_Allgather` call. The time taken to reset the buffer contents also affects the optimum value of M_T.

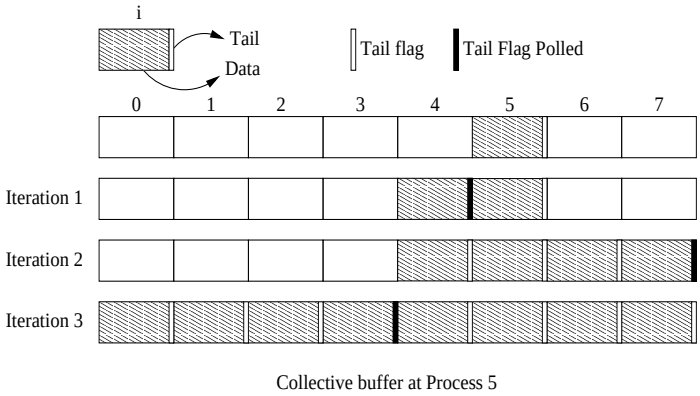


Fig. 3. Recursive Doubling: Copy-based approach RDMA buffer scheme

Back-to-back buffers: To allow back-to-back `MPI_Allgather` collectives, we provide three contiguous collective buffers which are used in a cyclic manner. It is to be noted that due to the copy-based buffering scheme used for small messages, two sets of contiguous buffers are not sufficient. If a process enters a `MPI_Allgather` call, we can be sure that no other process is writing into the next collective buffer. However, since in our design no process actually waits for all sends to complete from a specific collective buffer, some sends might still be pending. [1] Thus, we are required to have three sets of collective buffers. Since each collective buffer is 8KB (as mentioned before), the total buffer requirements for RDMA RD is 24KB.

Upon a call to RDMA based `MPI_Allgather`, if it is determined that zero copy approach will be used (depending on message size during any step), then we must make sure that we wait for all the sends from the buffer to be complete. This is required since the message is not buffered, the application cannot be given control without ensuring that all sends from that buffer are indeed complete.

4.3 RDMA Ring for large messages

We implement the Ring algorithm for `MPI_Allgather` over RDMA only for large messages and large clusters. As observed in [19], large clusters may be better near-neighbor bandwidth. Under such scenarios, it is beneficial for `MPI_Allgather` to mainly communicate between neighbors. The Ring algorithm is ideal for such cases. Since we implement this algorithm for only large messages, we use a complete zero

[1] In [17], the authors propose two back-to-back buffers for the Direct Eager RDMA buffering scheme. In that case two buffers were enough because control reached the application only on completion of all sends.

copy approach here. The design in this case is much simpler. The benefit of the RDMA-based scheme comes from the fact that we have a single buffer registration and a single address exchange performed by each node instead of p registrations, and $(p-1)$ address exchanges in the point-to-point based design. We use this Ring algorithm for messages larger than 1 MB and process numbers greater than 32.

5 Experimental Evaluation

In this section, we evaluate the performance of our designs. We use two cluster configurations for our tests. The cluster descriptions are as follows:

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

We have integrated our RDMA based design in the MVAPICH [15] stack. Our RDMA implementation uses both Recursive Doubling and Ring algorithms as discussed in section 4. Since our implementation automatically switches between both algorithms based on the message size and the number of processes, we simply refer to the new design as "MVAPICH-RDMA". The current implementation of `MPI_Allgather` over point-to-point is referred to as "MVAPICH-P2P".

Our experiments are classified into two types. First, we demonstrate the latency of `MPI_Allgather` for varying message sizes and process numbers. Secondly, we investigate the performance of our RDMA based designs in the case where buffer reuse ratio is very low.

5.1 Latency benchmark for `MPI_Allgather`

In this experiment, we measure the basic latency of our `MPI_Allgather` implementation. All the processes are synchronized with a barrier and then `MPI_Allgather` is repeated 1000 times. For all the iterations, the same communication buffers are used. The time observed is averaged over all the iterations, and finally averaged over all processes. The results are shown in Figure 4 and 5 for Cluster A. Results for cluster B results are shown in Figure 6. The results from both Cluster A and B follow the same trends. The results are explained as follows:

Small Messages: We observe that for smaller messages, we can reduce the overhead. As described in section 3, the RDMA based design can avoid the various copy overheads in different layers of the MPI point-to-point implementation. In addition, it can avoid other software overheads too. The results indicate that latency can be reduced by 17%, 13% and 15% for 16 processes on Cluster A (Fig 4(a)), 32 processes on Cluster A (Fig 5(a)) and 16 processes on Cluster B (Fig 6(a))for 4 byte message size respectively.

Medium Messages For medium sized messages, the point-to-point based design required rendezvous address exchange for transferring messages at every step of the algorithm. However, for the RDMA based `MPI_Allgather`, no such exchange is required (section 4). We note from section 2.3, the number of steps is proportional to either $log(p)$ or $(p-1)$ (depending on message size). Thus, this cost of address exchanges increases with number of processes. Our RDMA based design is able to successfully avoid this increasing cost.

The results indicate that latency can be reduced by 23%, 30% and 37% for 16 processes on Cluster A (Fig 4(b)), 32 processes on Cluster A (Fig 5(b)) and 16 processes on Cluster B (Fig: 6(b))for 32 KB message size respectively.

Large Messages Large messages are also transferred using the same zero copy technique used for medium sized messages. Hence, the same address exchange cost can be saved (as described in the previous case). However, since the message sizes are large, the address exchange forms a lesser portion of the overall cost of `MPI_Allgather`. The results for large messages indicate that latency can be reduced by 7%, 6% and 21% for 16 processes on Cluster A (Fig 4(c)), 32 processes on Cluster A (Fig: 5(c)) and 16 processes on Cluster B (Fig 6(c)) for 256 KB message size respectively. The performance improvement for Cluster B is larger because PCI-Express has much higher bandwidth than PCI-X. Since Cluster B is equipped with PCI-Express, removing the synchronization overhead (required for the rendezvous protocol) can lead to better bandwidth utilization for larger message sizes.

Scalability We plot the `MPI_Allgather` latency numbers with varying process counts, for a fixed message size to see the impact of RDMA design on scalability. Figure 7(a) shows the results for 32 KB message size. We observe that as the number of processes increase, the gap between the point-to-point implementation and RDMA design increases. This is due to the fact that the RDMA design eliminates the need for address exchange (which increases as the number of processes).

5.2 `MPI_Allgather` latency with no buffer reuse

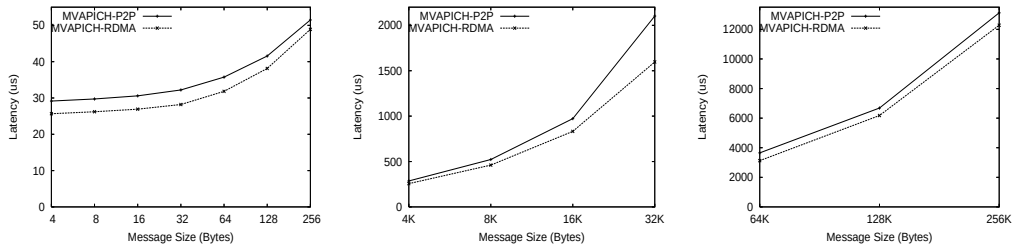
In the above experiment, we measured the latency of `MPI_Allgather` when utilizing the same communication buffers for a large number of iterations. The cost of registration was thus amortized over all the iterations by the registration cache maintained by MVAPICH. As mentioned in section 3, InfiniBand requires communication buffers to be registered. The cost of registration is quite high. Figure 7(b) shows the cost of registration on Cluster A for varying buffer sizes (in terms of number of pages).

However, it is not necessary that all MPI applications will always reuse their buffers. In the case where applications use `MPI_Allgather` with different buffers, the point-to-point based design will be forced to register the buffers separately, thus incurring high cost.

In this experiment, we conduct the same latency test (as mentioned in previous section), but the buffers used for each iteration are different. Thus, there is no buffer reuse. Figures 8 and 9 show the results for Clusters A and B respectively.

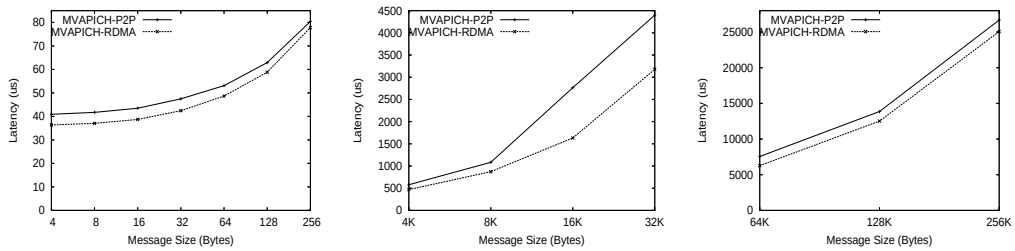
We observe from Figure 8 that there is a sudden jump in the point-to-point based `MPI_Allgather` for message size 16KB. This is because at this time, the Ring algorithm is being used. From section 2.3, we note that this would lead to p registrations instead of $log(p)$ in Recursive doubling.

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



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

Fig. 4. MPI_Allgather performance on 16 Processes (Cluster A)

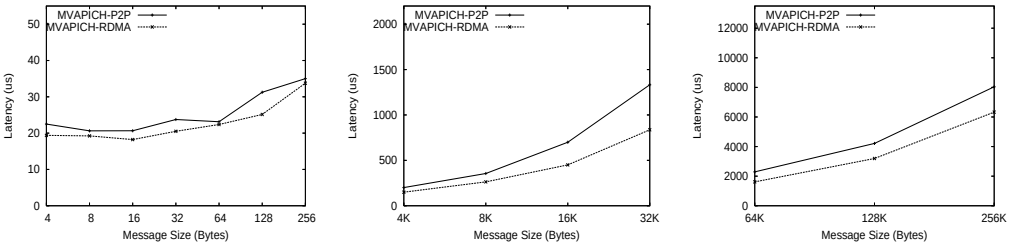


(a) Small Messages (b) Medium Messages (c) Large Messages

Fig. 5. MPI_Allgather performance on 32 Processes (Cluster A)

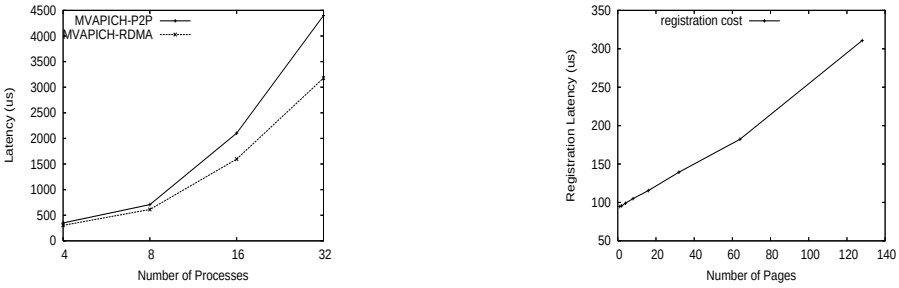
6 Related Work

Recently, a lot of work has been done to improve the performance of collective operations in MPI. In [19] the authors have implemented several well known collective algorithms over the MPI point-to-point primitives. In [9] and [17] and [12] the authors have shown the benefits of using RDMA for `MPI_Barrier`, `MPI_Alltoall` and `MPI_Allreduce` collective primitives respectively. In addition, researchers have been focusing on framework for non-blocking collective communication [8]. However



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

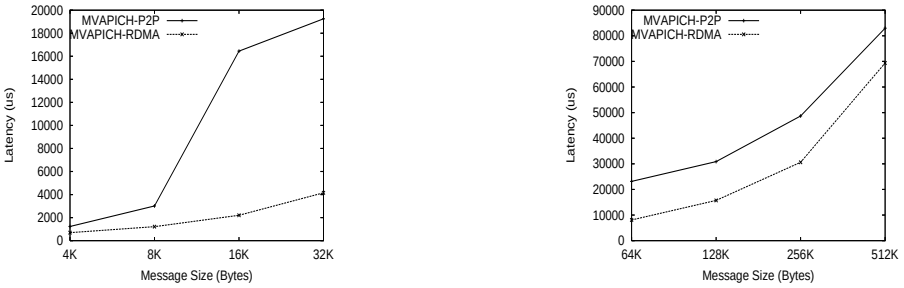
Fig. 6. MPI_Allgather performance on 16 Processes (Cluster B)



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

(b) Cost of Registration

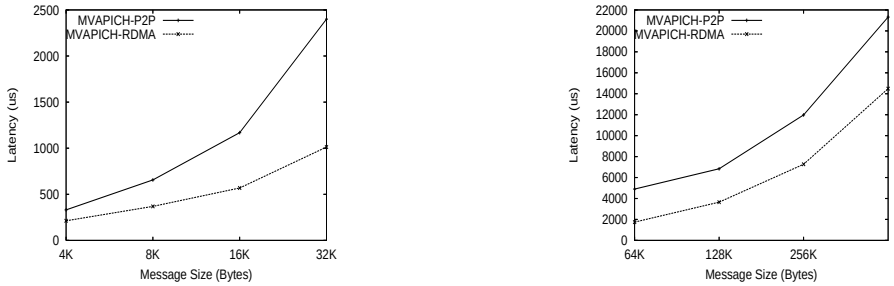
Fig. 7. Scalability and Registration Cost on Cluster A



(a) Medium Messages

(b) Large Messages

Fig. 8. No buffer reuse performance results for 32 processes on Cluster A



(a) Medium Messages

(b) Large Messages

Fig. 9. No buffer reuse performance results for 16 processes on Cluster B

our work is different from the above since our work mainly focuses on the All-to-all broadcast of messages using RDMA feature and intelligently choosing the thresholds for copy and zero-copy approaches. Our solution is also according to the MPI specification which require blocking collectives.

7 Conclusion and Future Work

In this paper, we propose a novel RDMA based design for the All-to-all Broadcast collective operation. Our design reduces software overhead, copy costs, protocol handshake – all required by the implementation of collectives over MPI point-to-point. Our design can dynamically choose the optimal threshold for performing zero copy while the collective progresses. Our design can significantly reduce costs by reducing the need for multiple buffer registrations. Performance evaluation of our designs reveals that the latency of `MPI_Allgather` can be reduced by 30% for 32 processes and a message size of 32 KB. Additionally, the latency can be improved by a factor of 4.75 under no buffer reuse conditions for the same process count and message size.

Our future work plan is as follows: Our RDMA based design of All-to-all broadcast can be internally utilized to perform several tasks for other MPI collectives which require virtual buffer address distribution at the collective start (e.g. RDMA based All-to-all personalized communication [17]). We want to evaluate the impact of this design on a much larger scale cluster. Further, we would like to extend this design to have an optimal algorithm for clusters of SMPs (combining shared memory and RDMA based design).

8 Acknowledgments

We would like to thank Intel for providing us access to their 16-node PCI-Express cluster. We would also like to thank Abhinav Vishnu, Wei Huang, Lei Chai, Gopal Santhanaraman, Pavan Balaji, Karthikeyan Vaidyanathan, Sundeep Narravula and Weikuan Yu for their feedback on technical issues.

9 Software Distribution

As indicated earlier, the open-source MVAPICH [15] software is currently being used by more than 210 organizations world-wide. The latest release is 0.9.5. The proposed RDMA based All-to-all broadcast solution will be available in the 0.9.6 release.

References

1. MPI: A Message-Passing Interface Standard. http://www.mpi-forum.org/docs/mpi-11-html/mpi-report.html.
2. D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, D. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber, H. D. Simon, V. Venkatakrishnan, and S. K. Weeratunga. The NAS parallel benchmarks. volume 5, pages 63–73, Fall 1991.
3. J. Bruck, C.-T. Ho, S. Kipnis, E. Upfal, and D. Weathersby. Efficient Algorithms for All-to-All Communications in Multiport Message-Passing Systems. *IEEE Transactions in Parallel and Distributed Systems*, 8(11):1143–1156, November 1997.
4. J. Duato, S. Yalamanchili, and L. Ni. *Interconnection Networks: An Engineering Approach*. The IEEE Computer Society Press, 1997.
5. W. Gropp, E. Lusk, N. Doss, and A. Skjellum. A High-Performance, Portable Implementation of the MPI, Message Passing Interface Standard. Technical report, Argonne National Laboratory and Mississippi State University, 1996.
6. InfiniBand Trade Association. InfiniBand Architecture Specification, Volume 1, Release 1.2. http://www.infinibandta.com.
7. InfiniBand Trade Association. InfiniBand Trade Association. http://www.infinibandta.com.
8. L. V. Kale, S. Kumar, and K. Vardarajan. A Framework for Collective Personalized Communication. In *International Parallel and Distributed Processing Symposium*, 2003.
9. S. P. Kini, J. Liu, J. Wu, P. Wyckoff, and D. K. Panda. Fast and Scalable Barrier using RDMA and Multicast Mechanisms for InfiniBand-Based Clusters. In *Euro PVM/MPI*, 2003.
10. Lawrence Berkeley National Laboratory. MVICH: MPI for Virtual Interface Architecture. http://www.nersc.gov/research/FTG/mvich/index.html, August 2001.
11. Lawrence Livermore National Laboratory. The ASCI Purple Benchmarks. http://www.llnl.gov/asci/platforms/purple/rfp/benchmarks/.
12. A. Mamidala, J. Liu, and D. K. Panda. Efficient Barrier and Allreduce on IBA clusters using hardware multicast and adaptive algorithms. In *IEEE Cluster Computing*, 2004.
13. Message Passing Interface Forum. *MPI-2: Extensions to the Message-Passing Interface*, Jul 1997.
14. Myricom Inc. Portable MPI Model Implementation over GM, March 2004.
15. Network-Based Computing Laboratory. MPI over InfiniBand Project. http://nowlab.cis.ohio-state.edu/projects/mpi-iba/.
16. Quadrics. MPICH-QsNet. http://www.quadrics.com.
17. S. Sur, H.-W. Jin, and D. K. Panda. Efficient and Scalable All-to-All Exchange for InfiniBand-based Clusters. In *International Conference on Parallel Processing (ICPP)*, 2004.
18. R. Thakur, R. Rabenseifner, and W. Gropp. Optimization of Collective communication operations in MPICH. *Int'l Journal of High Performance Computing Applications*, 19(1):49–66, Spring 2005.
19. Rajeev Thakur and William Gropp. Improving the Performance of Collective Operations in MPICH. In *Euro PVM/MPI 2003*, 2003.