Efficient Shared Memory and RDMA based design for MPI_Allgather over InfiniBand*

Amith R. Mamidala, Abhinav Vishnu, and Dhabaleswar K. Panda

Department of Computer Science and Engineering
The Ohio State University
{mamidala, vishnu, panda}@cse.ohio-state.edu

Abstract. MPI_Allgather is an important collective operation which is used in applications such as matrix multiplication and in basic linear algebra operations. With the next generation systems going multi-core, the clusters deployed would enable a high process count per node. The traditional implementations of Allgather use two separate channels, namely network channel for communication across the nodes and shared memory channel for intra-node communication. An important drawback of this approach is the lack of sharing of communication buffers across these channels. This results in extra copying of data within a node yielding sub-optimal performance. This is true especially for a collective involving large number of processes with a high process density per node. In the approach proposed in the paper, we propose a solution which eliminates the extra copy costs by sharing the communication buffers for both intra and inter node communication. Further, we optimize the performance by allowing overlap of network operations with intra-node shared memory copies. On a 32, 2-way node cluster, we observe an improvement upto a factor of two for MPI_Allgather compared to the original implementation. Also, we observe overlap benefits upto 43% for 32x2 process configuration.
Keywords: MPI, MPI_Allgather, RDMA, Shared Memory

1 Introduction

Clusters of commodity PCs are being increasingly deployed for high-end computing owing to their high performance-to-price ratios. Infact, many top 500 supercomputers are large scale clusters. These high-end systems are typically equipped with more than one processor per node such as a 2-way/4-way/8-way SMP or NUMA architecture. Also, the next generation systems feature multi-core support enabling more processes to run per processor. Already systems with dual-core and quad-core support have entered the high performance computing arena. This is expected to increase in future with even higher multi-cores being inducted to build ultra-scale clusters.

Message Passing Interface (MPI) [9] has evolved as the de-facto programming model for writing parallel applications. MPI provides many point-to-point and collective primitives which can be leveraged by these applications. Many parallel applications [7] employ these collective operations. MPI_Allgather is one such important operation which is used in applications involving matrix multiplication, solving differential equations

* This research is supported in part by Department of Energy's Grant #DE-FC02-01ER25506; National Science Foundation's grants #CCR-0204429, #CCR-0311542 and #CNS-0403342; grants from Intel and Mellanox; and equipment donations from Intel, Mellanox, AMD, Apple, Advanced Clustering and Sun Microsystems.

and in basic linear algebra operations. Thus, optimizing the performance of this operation on the emerging next generation cluster architecture presents an important problem.

Recently, InfiniBand has emerged as one of the leaders in the high performance networking domain [4]. It provides RDMA which enables a process to directly write data to a remote process's address space. We have shown the benefits of using this feature for various collective operations such as MPI_Barrier, MPI_Allgather, MPI_AlltoAll [5] [8] [11] [10]. But, these approaches are optimal for one process running per node. With the next-generation systems going multi-core, it is essential to choose the fastest communication methods offered by the underlying system and network interconnect for efficient collective operations. In the existing approaches, collective communication is performed by utilizing two different channels, shared memory channel for intra-node communication and network channel for communication across the nodes. The major drawback of this approach is that as these two channels do not share the communication buffers, multiple copies are involved in the whole operation. This significantly degrades the performance on large clusters especially with multiple processes running on a single node. Also, since these channel do not have common buffers, overlapping communication across shared memory and network is difficult to accomplish.

In this paper, we propose a combined shared memory and RDMA based design which overcomes the problem outlined above. The copy costs are eliminated in our design by allowing the data buffers to be shared for both communication within the node and across the nodes. Our design extends the traditional recursive doubling algorithm for Allgather to accommodate more processes per node. Also, since the communication buffers in our design are shared, there is a benefit of overlapping of intra- and inter-node communication. We have implemented our designs and integrated them into MVAPICH [6] which is a popular MPI implementation for InfiniBand used by more than 375 organizations worldwide. We have evaluated our designs on two different cluster configurations. For a 32x2 configuration, our design improves the latency of the MPI_Allgather by a factor of two. Further, we observe that the overlapping network and shared memory communication improves the performance upto 43% in the latency of MPI_Allgather.

The rest of the paper is organized in the following way. In Section 2, we provide the background of our work. In Section 3, we explain the motivation for our scheme. In Section 4, we discuss detailed design issues. We evaluate our designs in Section 5 and talk about the related work in Section 6. Conclusions and Future work are presented in Section 7.

2 Background

2.1 Recursive Doubling Algorithm for Allgather

In this algorithm, a pair of processes exchange data for every step. The total number of steps in the algorithm is of the order of $log(p)$ where p is the number of processes in the operation. Also, the data involved for each step doubles as the operation progresses, hence the name recursive doubling. 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. MPICH [3] [12] uses Recursive doubling algorithm for power of two and up to medium size messages. For non-power of two

processes, Bruck's Algorithm [2] is used for small messages. In this paper, we consider only the power of two case and hence we focus on the recursive doubling technique.

2.2 InfiniBand Overview

The InfiniBand Architecture [6] defines a switched network fabric for interconnecting processing and I/O nodes. 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. 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. 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.

2.3 Point-to-Point MPI operations in MVAPICH

The two main protocols used for MPI point-to-point primitives are the eager and rendezvous protocols. In the eager protocol, the message is copied into communication buffers at the sender and destination process before it is copied into the user buffer. These copies are not present if rendezvous protocol is used. However, in this case an extra handshake is required to exchange user buffer information for zero-copy of the message. In this paper we deal with small to medium messages which are sent using the eager protocol and thus copy operation is involved at both the sender and the receiver. For intra-node communication, a separate shared memory channel is used for communication. In MVAPICH, the shared memory channel involves each MPI process on a local node attaching itself to a shared memory region at the initialization phase. This shared memory region can then be used amongst the local processes to exchange messages and other control information. Each pair of the local processes has its own send and receive queues. Small and medium messages are sent eagerly, where as a packetization approach is used for large messages.

3 Motivation

The traditional implementation of MPI_Allgather for multi-way SMP-based clusters uses MPI point-to-point operations. Depending on the pair of processes communicating, these operations use either the network channel or the shared memory channel for communication. Consider a scenario where eight processes are involved in Allgather with two processes per node as shown in Figure 1. The total number of steps involved in the operation is 3 which is $log(8)$.

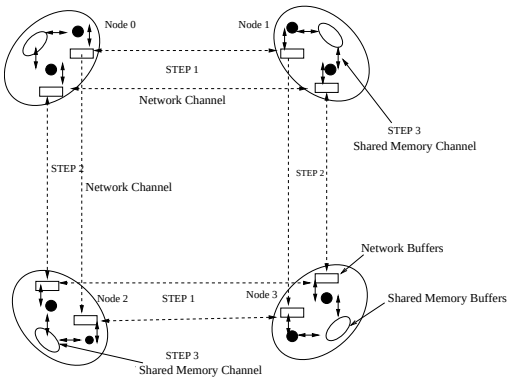

Fig. 1. Separate communication buffers

Depending on which pair is communicating at a step, the communication either proceeds over one of the two channels. Also, these two channels are designed separately and consequently do not allow sharing of buffers across the channels.

In the example considered, the first two steps involve the inter-node communication over the network channel. The third step involves the shared memory channel. Please note that we have taken this sequence of operations to illustrate the main idea. The network and shared memory operations can be scheduled in a different order depending on how the processes are launched on these nodes. As seen from the figure, separate sets of pinned buffers are associated with each channel for transmitting the data. As a result, though all the data for a given step has arrived at a node from other processes, it cannot be copied to every process local to the node. This is important in Allgather which involves an All-to-All broadcast of data. The reason why the data cannot be copied is because the network buffers are exclusive to a network channel and only the process communicating via this channel can access these buffers. Hence, a separate shared memory channel is needed resulting in extra copying of the data. Also the total amount of data exchanged increases linearly with the total number of processes participating in the operation. Thus, on a large cluster with more than one process per node, the copy costs play a dominant role degrading the performance of the collective operation. Another aspect to be taken into consideration is that since the buffers are not shared across the channels, overlapping shared memory and network communication becomes difficult to do. This further degrades the performance of this all-to-all operation.

This leads us to the following two questions:

1) *What mechanisms are needed to optimize MPI_Allgather for the emerging multi-core/multi-way InfiniBand Clusters?*

2) *How can we schedule the operations so as to easily allow overlap of network and shared memory operations?*

We address these questions in this paper.

4 Design and Implementation

The basic idea used in our approach is to use a common memory segment both for intra and network communication. This memory segment is shared across all the processes local to the node. Further, this segment is pinned so that it can be accessed directly by the NIC for the network operation. We now outline the main steps involved in our approach.

Our Approach: We extend the recursive doubling algorithm discussed earlier to be performed across the nodes rather than across the processes. In this fashion, a single message is exchanged per a pair of nodes irrespective of how many processes are scheduled on a node. This is accomplished by making all the local processes write their data into the shared memory segment in the initial step. This is the step 0 as shown in the Figure 2. Once all the processes have written the data into this buffer, the data exchange starts over the network. In the first step, node pairs 0, 1 and 2, 3 exchange the data. Note that the data exchanged in this step is one fourth the size of the total data. After this step, the second step as shown in the Figure 3 begins. The size of the data exchanged in this step is doubled as seen from the figure. The pairs which are involved in this exchange are now 0, 2 and 1, 3. Once this step is completed, each node has the data from all the processes. In the final step, which is the step four, the data is copied out of the shared memory segment.

As can be seen from the above example, in our approach the data is exchanged across the nodes in a recursive pair-wise fashion with a single data transfer operation between each pair of nodes. The number of steps would be equal to $log(n)$ where n is

the number of nodes involved in the operation. In the example considered, the number of steps is $log(4)$ which is two. Note that by providing a common set of buffers for both network and intra-node data transfers, we eliminate the extra copying that would otherwise occur.

Overlap benefits: The main benefits of having a shared buffer is the potential of overlap between the network operations and the memory copy operations. By referring to the same Figures 2 and 3, it can be observed that the data arrived in step 1 of the operation can be copied to the processes' buffers concurrently with network operation in step 2. Thus, we need not wait till all the network operations are completed before the data is copied out of the shared memory segment. For a large scale cluster, this benefit is significant as both the size of the data involved is large and also there are more steps involved in the algorithm.

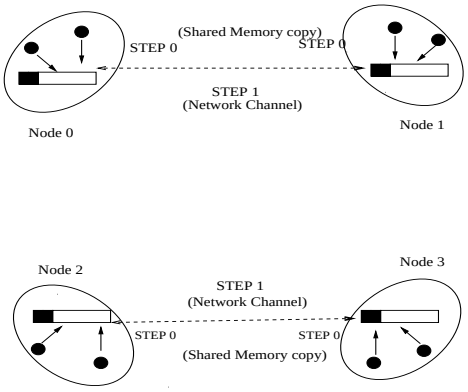

Fig. 2. Steps 0,1

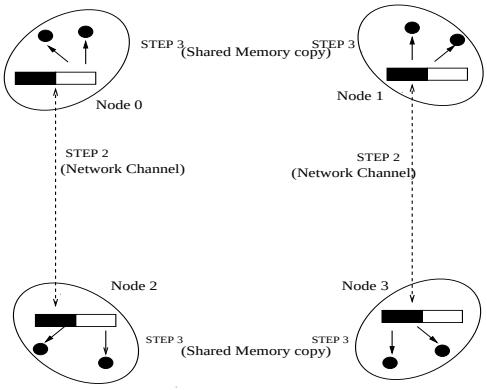

Fig. 3. Steps 2,3

Implementation Details: The initial implementation step in our approach is creating a shared memory segment per node. This is done by making all the processes local to a node do a mmap of a shared file. After this step, this shared segment is pinned so that data can be accessed directly by the NIC for the network operation. In our design, the shared buffer is pinned by all the processes. This enables all the processes to issue network operations from this memory segment. RDMA is used for network data transfers as it is proven to be an efficient method for inter-node communication. In our implementation, we let one given process issue the network operations from a node. This can be easily accomplished as the processes have local ranks ranging from 0 to p-1 where p is the total number of processes per node. We choose the process with local rank 0 to issue network operations. Note that the addresses of this memory segment are exchanged before the Allgather is initiated. The data notification is done by doing a RDMA write of a one byte flag. These flags are also shared within a node and thus all the processes local to the node can poll for data arrival. This is useful for achieving overlap between network and shared memory copy operations. For synchronizing between the processes within a node another separate set of flags are used.

5 Performance Evaluation

In this section we compare the performance of the new scheme proposed in the paper with the already existing approach. The comparison is made by measuring the *Allgather latency* for the two schemes across different message sizes and for two different cluster

configurations. The test was conducted for 1000 iterations for each message size. The abbreviations used for the comparison are as follows:

- new: The new shared-memory and RDMA based solution proposed in the paper.
- original: The original algorithm using MPI point-to-point operations

5.1 Experimental Testbed

We have carried tests on two different clusters:

1) Cluster A: Each node in this testbed has dual Opteron 2.4 GHz processors, 1024 KB L2 cache. They are equipped with MT25204 InfiniBand HCAs with PCI-Express interfaces.

2) Cluster B: Each node in this cluster is a Xeon 2.66 GHz processor with 512 KB L2 cache. Each node is connected with MT23108 InfiniBand HCA with PCI-X interface.

5.2 Latency of MPI_Allgather

As the results indicate our approach outperforms the original approach for the different cluster configurations considered. For Cluster A we observe benefits upto a factor of 1.47 and 1.39 for 32 and 64 processes as indicated by Figures 4 and 5 respectively. On cluster B, we observe an improvement by a factor of 1.97 and 1.82 for the considered configurations, 16x2, 32x2. These are shown in Figures 6 and 7 respectively.

We have also measured the impact of overlap of network operations and shared memory communication on these clusters. The non-overlap approach is implemented by making the processes copy the data from the shared buffers at the end after the network operations are completed. But, for the overlap case the processes copy the data as soon as it arrives and concurrently issue network operations. This is the approach taken in this paper. With the shared buffer RDMA design proposed the overlap improves the performance of the collective upto 30% for Cluster A and 43% for Cluster B as shown in the Figures 8 and 9.

6 Related Work

Utilizing shared memory for implementing collective communication has been a well studied problem in the past. In [13], the authors proposed to use remote memory operations across the cluster and shared memory within the cluster to develop efficient collective operations. They apply their solutions to Reduce, Bcast and Allreduce operations on IBM SP systems. In our approach we consider a different collective Allgather which has different communication pattern and present the results on commodity clusters. In [1], the authors implement collective operations over Sun systems. In [14], the authors improve the performance of send and recv operations over shared memory and also apply the techniques for group data movement. We have also designed and implemented collectives, MPI_Barrier, MPI_AlltoAll, MPI_Allgather, [5] [8] [11] [10]based on RDMA. However, these collectives are optimized for a single process running on a node.

7 Conclusions and Future Work

MPI_Allgather is an important collective operation which is used in applications such as matrix multiplication and in basic linear algebra operations. The next generation systems feature multi-core architecture enabling a high process count per node. The

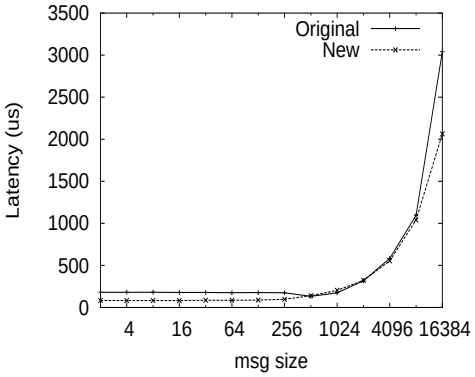

Fig. 4. Cluster A:(16x2)

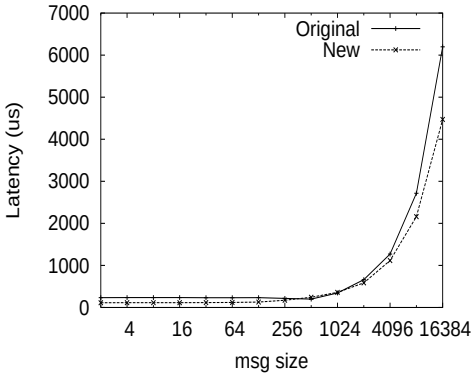

Fig. 5. Cluster A:(32x2)

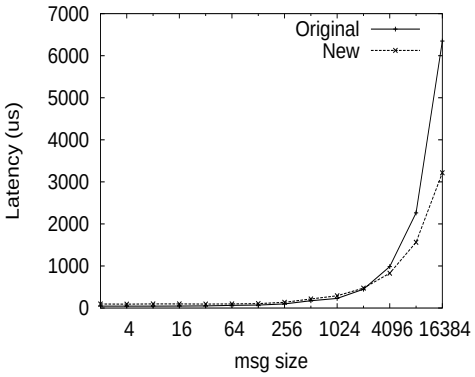

Fig. 6. Cluster B:(16x2)

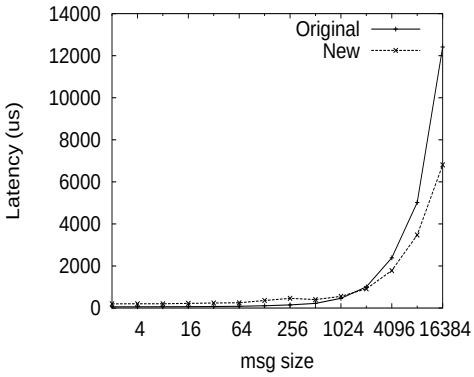

Fig. 7. Cluster B:(32x2)

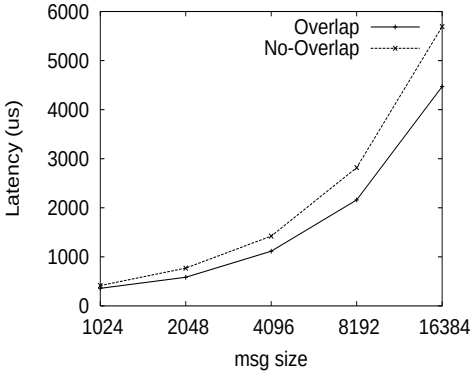

Fig. 8. Cluster A:(32x2)

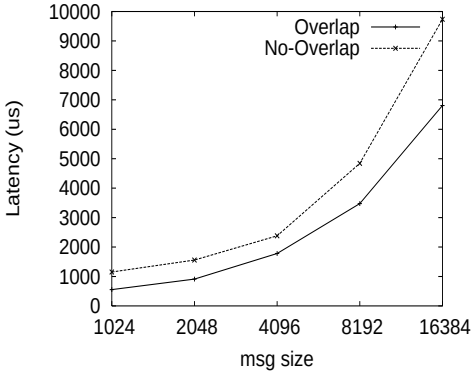

Fig. 9. Cluster B:(32x2)

traditional implementations of Allgather use two separate channels, namely network channel for communication across the nodes and shared memory channel for intra-node communication. Since there is no buffer sharing across these channels, the performance achieved is sub-optimal due to the extra copying of data within a node. This is true especially for a collective involving large number of processes with a high process density per node. In the approach proposed in this paper, we eliminate the extra copy costs by sharing the communication buffers for both intra and inter node communication. Also, we optimize the performance by allowing overlap of network operations with intra-node shared memory copies. On a 32, 2-way node cluster, we observe an improvement upto a factor of two for MPI_Allgather compared to the original implementation. We also observe overlap benefits upto 43% for 32x2 process configuration. For our future work, we plan to evaluate our design with multi-core enabled clusters and also study the application-level impact.

References

1. M Bernaschi and G Richelli. Mpi collective communication operations on large shared memory systems. In *Parallel and Distributed Processing, 2001. Proceedings. Ninth Euromicro Workshop*, 2001.
2. 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.
3. W. Gropp, E. Lusk, N. Doss, and A. Skjellum. A High-Performance, Portable Implementation of the MPI Message Passing Interface Standard. *Parallel Computing*, 22(6):789–828, 1996.
4. InfiniBand Trade Association. InfiniBand Architecture Specification, Release 1.1. http://www.infinibandta.org, October 2004.
5. Sushmitha P. Kini, Jiuxing Liu, Jiesheng Wu, Pete Wyckoff, and Dhabaleswar K. Panda. Fast and Scalable Barrier using RDMA and Multicast Mechanisms for InfiniBand-Based Clusters. In *EuroPVM/MPI*, Oct. 2003.
6. Jiuxing Liu, Jiesheng Wu, Sushmitha P. Kinis, Darius Buntinas, Weikuan Yu, Balasubraman Chandrasekaran, Ranjit Noronha, Pete Wyckoff, and Dhabaleswar K. Panda. MPI over InfiniBand: Early Experiences. Technical Report, OSU-CISRC-10/02-TR25, Computer and Information Science, the Ohio State University, January 2003.
7. NASA. NAS Parallel Benchmarks. http://www.nas.nasa.gov/Software/NPB/.
8. Amith R.Mamidala, Jiuxing Liu, and Dhabaleswar K. panda. Efficient Barrier and Allreduce InfiniBand Clusters using Hardware Multicast and Adaptive Algorithms . In *Proceedings of Cluster Computing*, 2004.
9. Marc Snir, Steve Otto, Steve Huss-Lederman, David Walker, and Jack Dongarra. *MPI–The Complete Reference. Volume 1 - The MPI-1 Core, 2nd edition.* The MIT Press, 1998.
10. S. Sur, U.K.R. Bondhugula, A.R. Mamidala, H.-W. Jin, and D. K. Panda. High performance RDMA based All-to-All Broadcast for InfiniBand Clusters. In *(HiPC)*, 2005.
11. 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.
12. 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.
13. V Tipparaju, J Nieplocha, and D K Panda. Fast collective operations using shared and remote memory access protocols on clusters. In *International Parallel and Distributed Processing Symposium, 2003*, 2003.
14. Meng-Shiou Wu, R A Kendall, and K Wright. Optimizing collective communications on smp clusters. In *ICPP 2005*, 2005.