Scalable Collective Communication for Next-Generation Multicore Clusters with InfiniBand

AMITH R. MAMIDALA, DEBRAJ DE, ABHINAV VISHNU, SUNDEEP NARRAVULA, DHABALESWAR K. PANDA

Technical Report
OSU-CISRC-6/07-TR49

Abstract

The emergence of multicore systems poses several challenges for the next-generation cluster architecture. Due to the availability of multiple processing cores per node, more number of application processes can be run on each node. In this context, the scalability of the cluster communication middleware like MPI is important for the overall performance of large scale applications.

MPI collective operations are widely used by many application codes to get good efficiency and high speed-up in performance. It is imperative that these operations scale both with respect to the protocols used and the communication resources employed for these multicore systems. SMP-aware collective optimizations have been well-researched in the past. However, these have been done over customized MPPs such as the IBM-SP and Sun Starfire. Also, most of the work has been targeted at improving performance. But, with sizes of commodity multicore clusters reaching several thousands of nodes it is important to ensure high performance of these collectives and also maintain optimum resource consumption. Recently, InfiniBand has emerged as the main stream high performance interconnect for commodity clusters. One of the important forms of message transport used by InfiniBand is the Reliable Connection (RC) mode. With this mode of transport, two important issues arise which need detailed investigation to design a scalable and high performance collective communication system. Firstly, the impact of concurrent network transactions can be significant for collective communication especially for the all-to-all communication pattern. Secondly, as clusters span tens of thousands of nodes resource consumption plays a key role for ensuring scalability.

In this paper, we explore these issues and study the utility of shared memory to enhance several aspects relating to performance and resource consumption. Specifically, we study the impact of cutting down of network transactions for important collective operations such as MPI_Barrier and MPI_Allreduce. For MPI_Alltoall, we evaluate message aggregation technique and use analytical modeling to obtain valuable insights. We also explain the savings obtained in the connection resources while using shared memory based collective operations. We have evaluated our designs over four node quad-core Intel Xeon Clovertown systems. On this testbed, our results show that an improvement by a factor of three can be achieved for MPI_Barrier and MPI_Allreduce. For MPI_Alltoall, we observe a performance improvement of two orders of magnitude for message sizes up to 1024 bytes for the Clovertown system testbed.

1 Introduction

Clusters of commodity PCs are becoming ubiquitous in the high performance computing arena owing to their high performance-to-price ratios. Recently, the shift in the trend towards multicore computing is presenting several challenges to the system designers to effectively harness the capacity of such systems. With the number of cores steadily increasing per node, several open questions emerge pertaining to the scalability of the various aspects of system design. One of such challenges is the scalability of the communication sub-system both with respect to the communication protocols employed and the amount of resources consumed.

Message Passing Interface (MPI) [15] 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 the applications codes. Especially, many parallel applications [12] employ collective operations for their efficiency and ease of use. Some of the widely used collective operations are MPI_Allreduce, MPI_Reduce, MPI_Alltoall, MPI_Bcast and MPI_Barrier. The advent of multicore clusters enables more than one MPI process to be run on a node. In such scenarios, it is important that these collective operations perform optimally and consume less communication resources. Several researchers have proposed optimal algorithms for these operations for varying system sizes and architectures. One such class of algorithms is the SMP-aware collectives. Traditionally, these algorithms have been studied in the context of customized massive MPPs such as the IBM-SP and Sun Starfire [9, 22]. Also, most of the work has been targeted at improving performance. But, with sizes of commodity multicore clusters reaching several thousands of nodes it is important to ensure high performance of these collectives and also maintain optimum resource consumption.

Recently, InfiniBand(IBA) has emerged as one of the main stream high performance commodity cluster interconnect. IBA is increasingly being deployed to interconnect several hundreds of multicore systems owing to its high bandwidth and low latency capabilities. IBA provides for different classes of message transport, the two important ones being the Reliable Connection (RC) and the Unreliable Datagram (UD). In this paper, we focus on the RC mode of message transport and understand the implications of its usage on the important collective operations, MPI_Allreduce, MPI_Alltoall and MPI_Barrier. This is especially important in scenarios where multiple MPI processes are run per node stressing the communication interfaces. We first understand the impact of concurrent network transactions over the RC transport. Though generally true for many collective operations, this study is particularly important for personalized All-to-all communication. We use both experimental and analytical methods to evaluate this behavior. We then evaluate the shared memory based optimizations to minimize the number of network transactions. Particularly, we explore message aggregation techniques to cut-down network operations and improve network utilization. Further, we explain how significant savings of communication resources can be achieved by utilizing shared memory methods.

Specifically, our objectives in this paper are the following:

- Analyze the impact of concurrent network transactions which can potentially be large for multicore systems.
- Analytically model and understand message aggregation techniques over InfiniBand.
- Evaluate the savings in communication resources obtained by using shared memory communication methods.
- Evaluate the shared memory optimizations for important collective operations such as MPI_Allreduce, MPI_Barrier etc. for the next-generation multicore architectures and analyzing the switch-points for different algorithms.

We have implemented our designs and integrated them into MVAPICH [8] which is a popular MPI implementation for InfiniBand used by more than 455 organizations worldwide. We have evaluated our designs over the quad-core Intel Xeon Clovertown systems. Our results show that an improvement by a factor of three can be achieved for MPI_Barrier and MPI_Allreduce. For MPI_Alltoall, we observe high degradation in performance without using shared memory message aggregation techniques.

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 followed by the Analytical model in Section 5. We evaluate our designs in Section 6 and talk about the related work in Section 7. Conclusions and Future work are presented in Section 8.

2 Background

In this section we first provide an overview of InfiniBand. We then present the standard algorithms used for different collective operations.

2.1 InfiniBand Overview

The InfiniBand Architecture (IBA) [5] defines a switched network fabric for interconnecting processing nodes and I/O nodes. It provides a communication and management infrastructure for inter-processor communication and I/O. In an InfiniBand network, processing nodes and I/O nodes are connected to the fabric by Channel Adapters (CA). Host Channel Adapters (HCAs) sit on processing nodes. InfiniBand supports different classes of transport services. In current products, Reliable Connection (RC) service and Unreliable Datagram (UD) service are supported. To use RC service, first a connection needs to be established between the participating processes. Also, each connection consumes memory resources of the host.

2.2 Collective Algorithms

In this section we present the different algorithms used for MPI collectives. For very small messages, MPI_Alltoall uses the Bruck's Algorithm [3] which is a store and forward algorithm taking $log(n)$ steps. For small to medium messages, MPI_Alltoall uses a non-blocking algorithm [18] where each node sends messages to all other nodes. Messages are scattered so that not all messages are directed towards a node to avoid congestion and hot-spot effects. A simple rule could be a process sends the ith message to process whose rank is (my_rank+ i)$\%$ n. These are currently used in MPICH [4]. This algorithm works well for medium messages where the cost of store and forward is absent [18]. MPI_Barrier is implemented using a dissemination based algorithm which consists of $log(n)$ steps [20]. For MPI_Allreduce, two algorithms are used depending on the message sizes. For short messages, a pair-wise exchange algorithm is used. As described above, this also consists of $log(n)$ steps where the data between the pairs is reduced.

2.3 MVAPICH Overview

We now provide a high-level design overview of Point-to-Point and Collective Communication support in the MVAPICH stack. As shown in Figure 1, the software stack is composed of three main components: a. Point-to-Point operations, b. Point-to-Point based Collective operations and c. Optimized Collective Operations. These are explained briefly below.

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. 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.

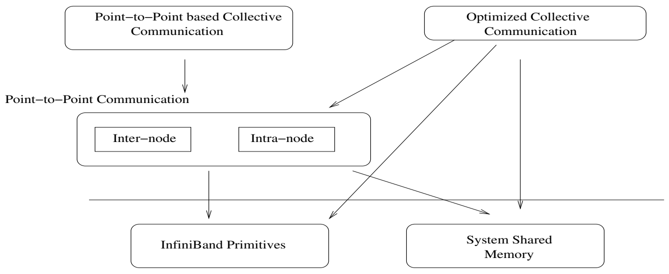

Point–to–Point based Collective Communication
Optimized Collective Communication
Point–to–Point Communication
Inter–node
Intra–node
InfiniBand Primitives
System Shared Memory

Figure 1. MVAPICH Design Overview

Point-to-Point based Collective operations: In MVAPICH, all the collective algorithms described above have default implementations over point-to-point operations. These are derived from MPICH implementation [20]. However, these algorithms were designed to be optimal for single process clusters. Also, since these algorithms are based on point-to-point operations, they cannot directly take advantage of the underlying network features. One such example is that of H/W Multicast.

Optimized Collective Operations: As discussed above, Collective Operations implemented directly over the point-to-point operations are not optimal. This is especially true for the emerging multicore InfiniBand clusters where more than one process can be launched on a single node. As the Figure 1 shows there are two ways of optimizing these operations for these cases. One way of doing this is to directly use the shared memory together with underlying network features. This is potentially useful for collective operations which involve large data transfers. Using shared memory together with RDMA primitive of InfiniBand leads to improved performance as shown by our earlier work [11]. Another alternative is to use the Point-to-Point based collective operations described above together with shared memory. For short messages where the copy costs are small, this approach does not place any significant overheads. Also, this approach is generic and can be easily integrated with the existing point-to-point implementations of collective operations. In this paper we study this approach for the collectives MPI_Allreduce, MPI_Barrier and MPI_Alltoall.

3 Motivation

The advent of multicore processors presents several interesting challenges for the emerging cluster architecture. Due to the availability of multiple processing cores per node, more number of application processes can be run on each node. This places critical demands on various system components such as the memory subsystem, communication resources etc. This is especially true with collective operations which concurrently stress the network interfaces.

Recently, InfiniBand(IBA) has emerged as one of the main stream high performance commodity cluster interconnect. IBA provides for different classes of message transport, the two important ones being the Reliable Connection (RC) and the Unreliable Datagram (UD). In this paper, we focus on the RC mode of message transport and understand the implications of its usage on the important collective operations, MPI_Allreduce, MPI_Alltoall and MPI_Barrier. As discussed earlier, these collective operations issue multiple concurrent network transactions stressing both the performance and resource consumption of the underlying interconnect.

Concurrent communication suffers from two major bottlenecks depending on the message sizes used. For small to medium messages, the contention is manifested in terms of start-up costs where as for large messages, bandwidth becomes the bottleneck. In this paper, we focus on the study of small to medium

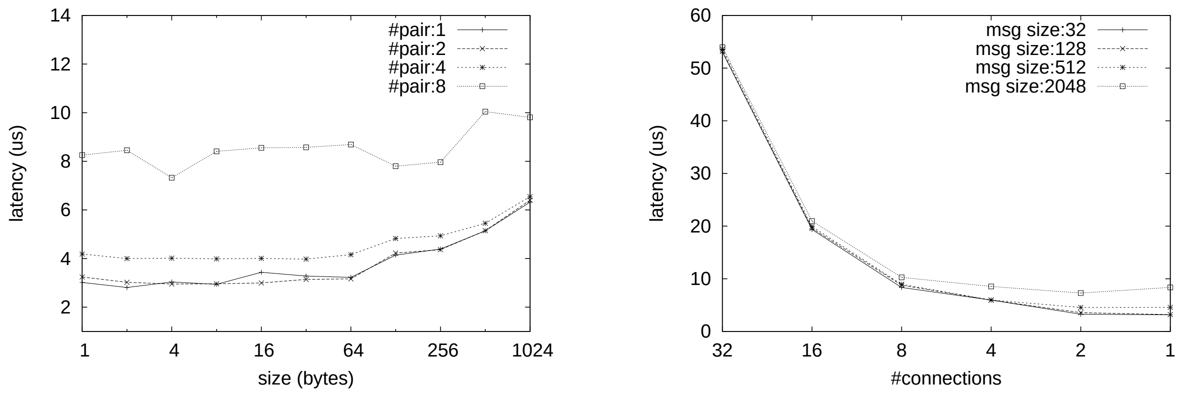


Figure 2. (a) Multi-pair latency (b) Scatter latency

messages whose latencies are dominated by the start-up costs. Figure 2(a) shows the effect of multiple network transactions occurring between varying pairs of processes. The experiment was conducted by concurrently running the ping-pong latency test between multiple pairs of processes. Two eight-core clovertown systems interconnected by InfiniBand adapters were used in the experiment. As shown in the figure, the latency shows marked degradation with the number of pairs rising to eight. For short message collective operations like MPI_Barrier, MPI_Allreduce and MPI_Alltoall, this can lead to degradation of the performance. This is because these collectives typically employ a pair-wise exchange algorithm for short messages. With multiple processes per node, the inter-node communication phases are prone to concurrent network accesses similar to the one shown in the Figure 2(a). Thus, it is important to avoid such hot-spots for these operations.

Apart from these performance bottlenecks, the amount of communication resources utilized in the collective operation critically affects the scalability of the overall system. For personalized collectives such as MPI_Alltoall, this is significant as multiple connections need to be established across the participating processes. Since each connection consumes communication resources, there is an $O(n)$ resource consumption per process hindering scalability of cluster middleware.

In this paper, we evaluate the shared memory mechanism to overcome the above challenges. Firstly, shared memory allows for message aggregation. The benefits of this approach can be seen from Figure 2(b). The figure shows the results of a scatter test where a constant message size is uniformly striped across varying number of connections. Thus, the fewer the connections used the larger is the message transmitted over the connection, As shown in the figure, scattering certain message sizes across smaller connections is beneficial rather than going over larger connections. However, these benefits are dependent on the characteristics of the underlying communication transport and the ability of the upper-layer communication protocols to fully saturate the pipeline depth. Secondly, shared memory based protocols can cut down the network transactions boosting performance. Finally, shared memory allows for reducing number of connections that need to be established. This leads to improved resource scalability.

Specifically, in this paper we aim to answer the following questions:

- What is the impact of utilizing shared memory in cutting down the number of network transactions for important collective operations such as MPI_Allreduce, MPI_Barrier and MPI_Alltoall?

- What are the benefits of aggregating messages across different connections for collective operations such as MPI_Alltoall?

- What analytical model can be derived for message aggregation over InfiniBand?
- What are the resource optimizations by utilizing shared memory for collective operations?

We explore these issues in the following sections of the paper.

4 Leader-based Collective Operations

In this section we provide the design and implementation details for the three collective operations studied in this paper: MPI_Barrier, MPI_Allreduce and MPI_Alltoall. The main idea in all of the three operations is using a single process per node called as the *Leader* process to do all the inter-node collective protocol processing and progress. As explained below, all the three collective operations are composed of three steps. Two of these are intra-node where only the local processes participate and one of them is inter-node where only the *Leader* process participates. The semantics of each of these steps varies with the type of collective operation.

$Collective$	$step:1$	$step:2$	$step:3$
MPI_Barrier	intra_Gather	inter_Barrier	intra_Bcast
MPI_Allreduce	intra_Reduce	inter_Allreduce	intra_Bcast
MPI_Alltoall	intra_Gather	inter_Alltoall	intra_Scatter

All the intra-node collective operations are performed via shared memory. This is done by making all the processes local to a node do a mmap of a shared file. However, prior to this step all the processes participating in the collective call have to be grouped to create a two level hierarchy of communicating groups. The first level consists of all the local processes to the node and the second level consists of all the *Leader* processes from each local group. Please note that all the processes in a particular group are ranked from 0 to $n-1$ where n is the size of the group. We choose the process with rank 0 of the local group to be the *Leader* process of that node. Apart from involving in the inter-node collective communication, the *Leader* process also performs the intra-node reductions in the case of MPI_Allreduce. Synchronization operations for these collective operations are performed using separate shared memory flags.

Packing/Unpacking for MPI_Alltoall: For shared memory based MPI_Alltoall operation, the intra_Gather step involves the packing of the data from all the local processes and the intra_Scatter step involves the unpacking of the data. We now explain the data layout for the buffers involved in packing/unpacking data. We assume a two node with two processes per node scenario to present the case.

The four processes P0, P1, P2 and P3 are grouped into two groups based on their locality. The two local groups being node_0:{P0, P1} and node_1:{P2, P3}. Each process has four data items to be sent to the other processes: P0<d0,d1,d2,d3>, P1<d0,d1,d2,d3>, P2<d0,d1,d2,d3> and P3<d0,d1,d2,d3>. For packing, data from each local process traveling to the same destination node has to be gathered to one buffer. In order to do this, the data is packed according to the local ranks of the source and destination processes. For the scenario considered, the data after packing is of the following form at each of the nodes:

node_0:{<d0,d1>P0, <d0,d1>P1}, {<d2,d3>P0, <d2,d3>P1}
node_1:{<d0,d1>P2, <d0,d1>P3}, {<d2,d3>P2, <d2,d3>P3}

5 Analytical Model

In this section we explain the analytical model used in this paper. This model is used for estimating the latencies of the scatter operation using the direct approach i.e. the root sends separate messages to all the participating processes one after another. This communication pattern forms the basis of the direct MPI_Alltoall algorithm as described in Section 2. The main objective of this model is the estimation of the cut-off point where aggregating messages is not beneficial.

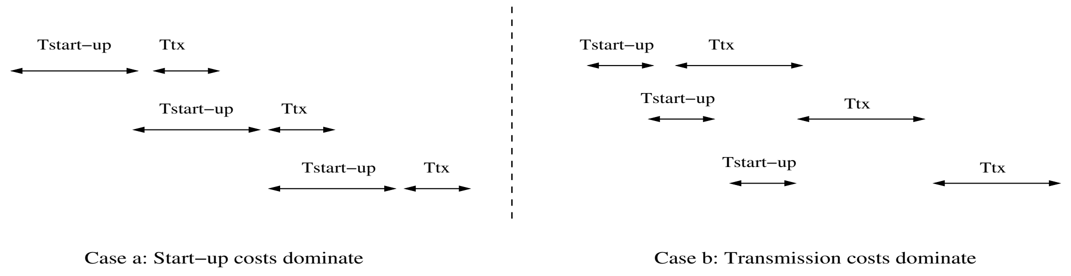


Figure 3. Timing behavior

The basic equation used in our modeling is:

$$T_{lat} = T_{serialization} + T_{constant} \tag{1}$$

The latency of the scatter operation is comprised of two factors, the cost of serialization of the operations which is dependent on the number of operations issued and an independent factor which remains constant. The per-operation serialization cost is once again determined by the size of the message used in the operation. Figure 3, depicts the two scenarios of small messages and large messages. As Case a in the figure illustrates, for small messages the latency of the operation is serialized by the start-up costs, shown in the figure as Tstart-up. Please note that Tstart-up is the cost of serializing the start-ups and not the exact cost of each start-up operation. To determine the total startup costs, an additional constant C is required. For large messages the latency of the operation is dominated by the serialization of the transmission time, Ttx. As a result, we have two equations describing the total transfer time of the operation over n processes depending on the size of the message employed.

$$T_{lat} = (n-1)T_{startup} + T_{transmission} + C \quad if \quad T_{startup} > T_{transmission} \tag{2}$$

$$T_{lat} = T_{startup} + (n-1)T_{transmission} \quad if \quad T_{startup} < T_{transmission} \tag{3}$$

Further $T_{transmission}$ can be further broken down into:

$$T_{transmission} = msg\_size \times \beta \tag{4}$$

where β is the per byte transmission cost.

The individual parameters for the equation are obtained by profiling scatter latencies of 1 byte and 16384 bytes of data across 32 processes. The experimental testbed details are provided in Section 6. The parameters obtained for the equations are: $T_{startup} = 2.098, \beta = 0.000935$ and $C = -10$. The negative value of C illustrates that the initial total time was over-estimated. This is because the serialization overhead is not present for all the transactions. Due to the availability of multiple DMA engines, there is some amount of concurrency in the processing of network transactions. Infact, for a 11 node scatter, the observed latency is 9.9 us. The model without the constant factor C gives 20 us which is around 10 us above the original value.

Figure 4 validates the model for different message sizes. As the figure illustrates, the model closely captures the behavior of scatter latencies.

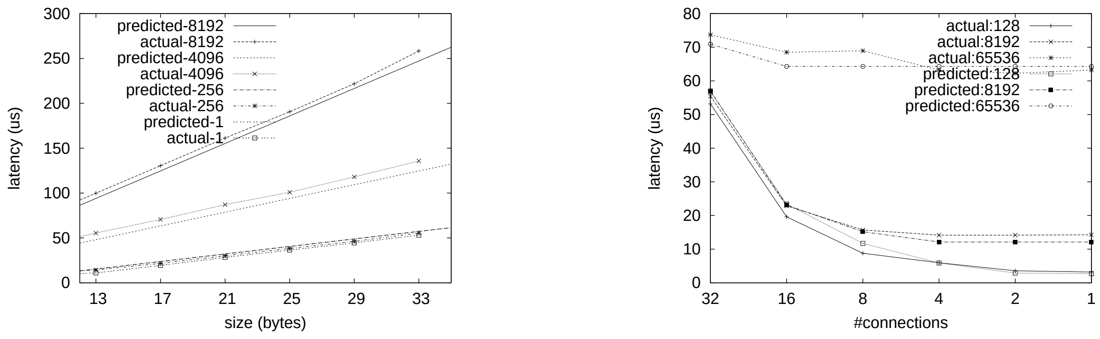


Figure 4. Model Verification:(a)Linear scatter (b) Message aggregation

From this model important conclusions can be drawn. As seen from the model, the benefits of message aggregation are not present if the serialization costs due to message transmission dominate the start-up costs. The message size at which this occurs can be obtained by determining the switch-point where the serialization cost of message transmission exceeds that of the start-up cost. This is shown in the following equation.

$$x \times \beta = T_{startup} \tag{5}$$

By substituting the different parameters, we obtain the values of 2243 as the value of x. Thus, after this point message aggregation does not yield benefits. This directly applies to MPI_Alltoall where each node does the scatter operation as discussed in this model. However, the exact latency of MPI_Alltoall is complicated to model compared to a single scatter operation. Because this involves several parameters which have to be modeled accurately. Some of these are the hot-spot effects due to network contention, memory contention for multi-core systems, caching effects etc.

6 Performance Evaluation

In this section we compare the performance of the shared memory enhancement of the three important collective operations: MPI_Allreduce, MPI_Barrier and MPI_Alltoall. We then evaluate the savings of connection resources with the leader-based collective operations.

6.1 Experimental Testbed

We have used three different clusters with varying characteristics to carry out in-depth performance evaluation and associated benefits of the proposed schemes.

Cluster A: Each node of the testbed is a duel Intel Xeon Clovertown processor with quad core. The nodes are interconnected by MT25208 HCAs in DDR mode. The operating system used is Redhat Linux

AS4. Our results were taken on four of such systems with a total of 32 cores. We have evaluated our designs on varying configurations such as 4x2(i.e. four processes on each of the two nodes), 4x4 and 4x8.

Cluster B: Each node of the testbed has two 3.6 GHz Intel processor and 2 GB main memory. The CPUs support the EM64T technology and run in 64 bit mode. The nodes are equipped with MT25208 HCAs with PCI Express interfaces. A 144-port DDR switch is used to connect all the nodes. The operating system used was RedHat Linux AS4. We have evaluated our designs on 16x2, 32x2 configurations.

6.2 MPI_Barrier Latency

Figure 5(a) shows the latency of the barrier operation on Cluster A. As shown in the figure, performance improvements up to a factor of 3 are observed for 32 processes. This significant speed-up in the performance is due to the lowering of network contention. As discussed in earlier sections 2, MPI_Barrier uses a pair-wise exchange algorithm. During the inter-node phases of the barrier, all the eight processes contend for the network interface thereby causing the performance to drop. Using shared memory alleviates this problem and causes only one network transaction between a pair of nodes. Similar trends can also be observed for Cluster B, Figure 5(b).

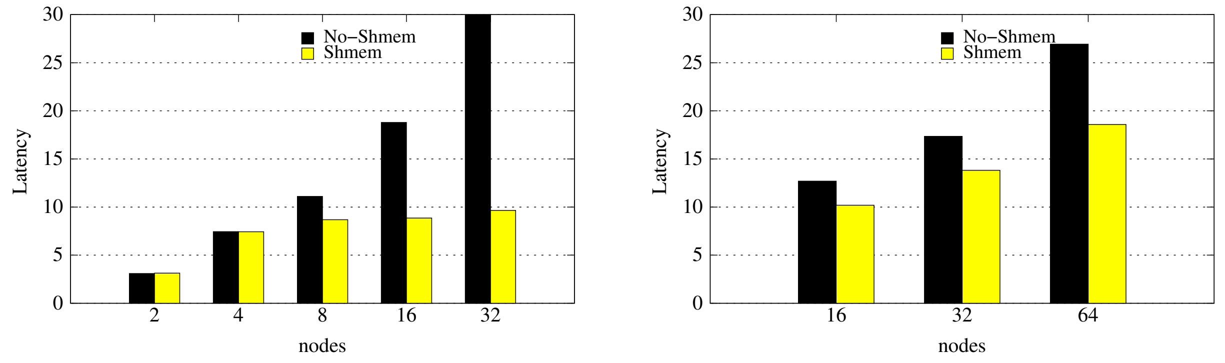


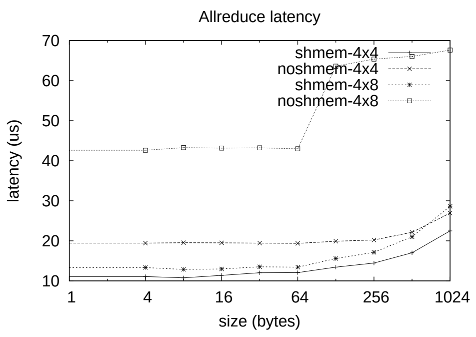
Figure 5. Barrier Latency:(a)Cluster A (b) Cluster B

6.3 MPI_Allreduce Latency

Figures 6 show the latency of the shared memory based allreduce on the clovertown multicore systems. As shown from the figure, the latency of the operation is significantly lowered upto a factor of three for short messages and up to 33% for large messages. The performance gain for short message allreduce is due to the cutting down of network transactions akin to barrier operation. However, similar performance gains are not observed for large data messages. This is because MPI_Allreduce uses a highly parallelized algorithm for large messages:Reduce-scatter followed by allgather. This is explained in detail in [20]. This offsets the performance gains obtained from following the leader-based design. Figure 7 shows the same behavior for a two-way SMP cluster.

6.4 MPI_Alltoall Latency

In this section we compare the different schemes of implementing MPI_Alltoall. These are outlined below:



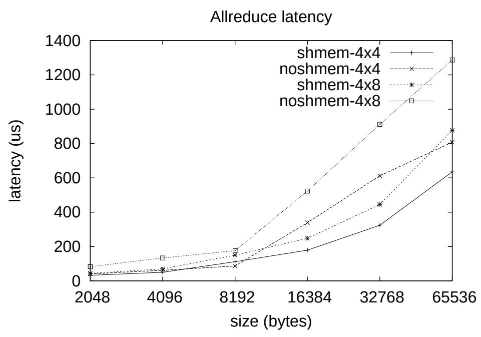
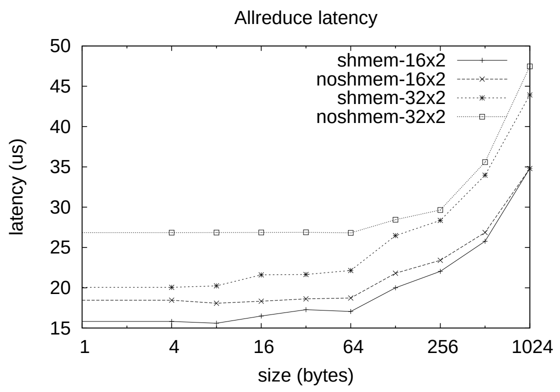


Figure 6. Allreduce Latency on Cluster A:(a)Small messages (b) Medium messages



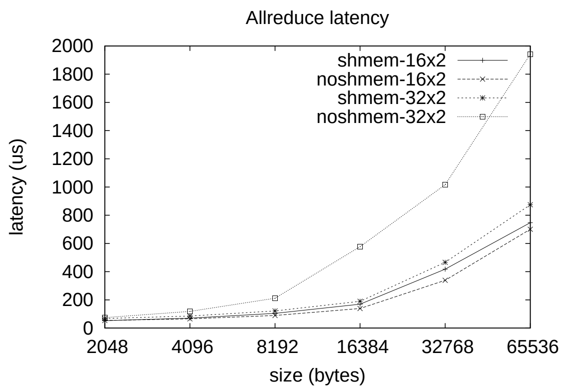


Figure 7. Allreduce Latency on Cluster B:(a)Small messages (b) Medium messages

- Direct without Packing: This is the default implementation of Direct Alltoall over the point-to-point operations
- Direct with Packing: This is the shared memory based Direct Alltoall implementation discussed in this paper
- Brucks without Packing: This is the default implementation of Bruck's Alltoall over the point-to-point operations
- Brucks with Packing: This is the shared memory based implementation of Alltoall using the Brucks algorithm

In the first part of this section we outline the impact of highly concurrent network operations using Direct MPI_Alltoall. We compare the two Direct schemes: with/without Packing and analyze the trends. We then compare the Brucks implementation with the Direct approach.

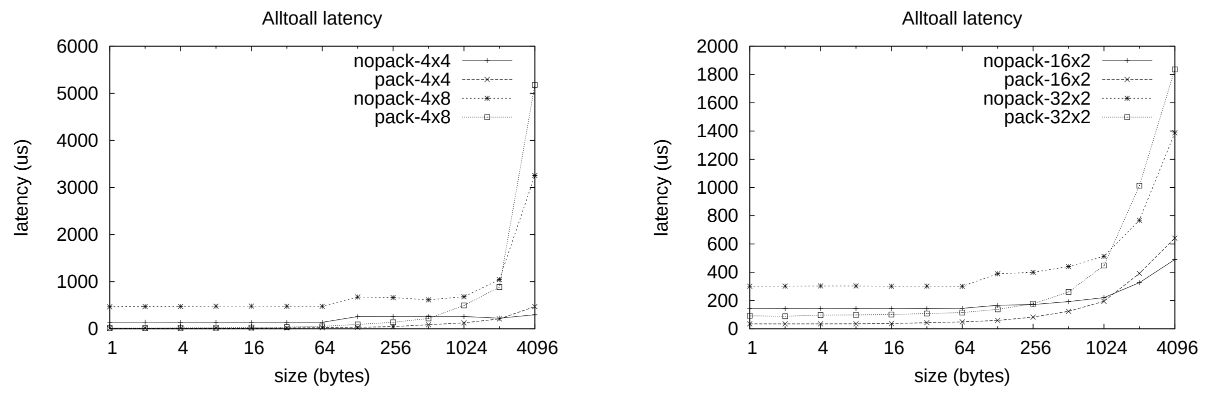


Figure 8. Direct Alltoall Latency :(a)Cluster A (b) Cluster B

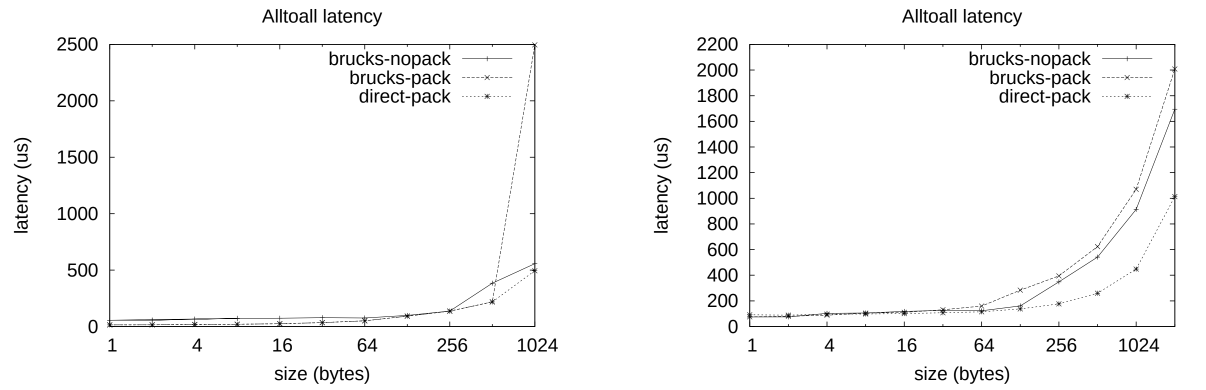


Figure 9. Alltoall Latency (a)Cluster A (b) Cluster B

Figure 8(a) shows the behavior of Direct Alltoall on four eight-core clovertown nodes. With Direct without packing, a total of (24*8) connections are established from a node with the same order of message exchanged between the nodes. This places heavy demands on the network interface as shown from the figure where the latencies are of two orders of magnitude higher than the packing based approaches where only one leader process communicates with the other four nodes. This drastically reduces the contention and provides very significant performance gains.

Also note that as shown from the analytical modeling, the packing and no packing show no benefit beyond 2243 bytes of message size. However, the exact cut-off point depends on several factors such as contention in the memory subsystem, hot-spot effects. This can be seen from Figure 8(b) where the cut-off point is between 1024 and 2048 where as in Figure 8(a) it is 2048.

Figures 9 shows the latencies of the three schemes in which Direct with packing shows better or equal performance to all the three other schemes. But Brucks with packing is worse compared to the other schemes in both the scenarios. This is because the cost of pack/unpack offsets the performance gains due to message aggregation.

6.5 Savings in Communication Resources

In this section we explain the savings obtained by doing shared memory based collectives. Please note that we have evaluated our implementation over the RC transport of InfiniBand. As discussed in earlier sections, an explicit connection needs to be established before communication. Each connection consumes memory resources for storing connection context data structures and communication buffers. These have been studied in [10, 7, 17, 19]. The following table gives the savings in total number of connections per node while using Shared memory(Shmem) alone: n refers to the total number of nodes and m is the number of cores on each node.

	Without Shmem	With Shmem	Savings
MPI_Alltoall	$O\left((n-1)m^2\right)$	$O\left(n-1\right)$	$O\left(m^2\right)$
MPI_Barrier	$O\left(mlog(n)\right)$	$O\left(log(n)\right)$	$O\left(m\right)$
MPI_Allreduce	$O\left(mlog(n)\right)$	$O\left(log(n)\right)$	$0\left(m\right)$

Please note that earlier studies [10] have shown the benefits of memory savings using the UD-transport of InfiniBand. However, these were studied over with single process/node scenarios. A thorough analysis of both the transports is needed for multicore systems to evaluate the optimal method both in terms of performance and memory consumption. Also, please note that utilizing the scheme mentioned in this paper will not be optimal for all messages in the case of MPI_Alltoall. Especially for large messages, the cost of packing/unpacking would be high. Thus, pipelining techniques have to be explored and the benefits studied. We intend to carry out these studies as part of our future work.

7 Related Work

Utilizing shared memory for implementing collective communication has been a well studied problem in the past. In [21], the authors propose using 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 operation. The authors evaluate their approach on IBM-SP systems. In this paper, we evaluate the SMP-based MPI_Allreduce, MPI_Barrier and MPI_Alltoall focusing on the impact of small message network contention on InfiniBand network interfaces. Especially techniques such as message aggregation have been analytically modeled and the switch-points identified. Message aggregation and the associated models for communication have been discussed by Panda et AL in [1]. These have been designed over wormhole routed using multi-destination message passing. These have been analyzed over torus topologies. In [2, 14], the authors implement collective operations over Sun systems. In [23], the authors improve the performance of send and recv operations over shared memory and also apply the techniques for group data movement. Collectives for single process per node over InfiniBand have been studied for MPI_Barrier, MPI_AlltoAll, MPI_Allgather, [6, 13, 18, 16] based on RDMA techniques. Shared memory collectives using RDMA have been explored in [11]. The benefits obtained here were due to savings from avoiding copy costs.

8 Conclusions and Future Work

The emergence of multicore clusters places several challenges for the next-generation cluster architecture. Due to the availability of multiple processing cores per node, more number of application processes can be run on each node. In this context, the scalability of the cluster communication middleware like MPI is important for the overall performance of large scale applications.

As the MPI collective operations are widely used by many applications it is imperative that these operations scale both with respect to the protocols used and the communication resources employed for multicore clusters. SMP-aware collective optimizations have been well-researched in the past. However, these have been done over customized MPPs such as the IBM-SP and Sun Starfire. InfiniBand has emerged as the main stream high performance interconnect for commodity clusters. One of the important forms of message transport used by InfiniBand is the Reliable Connection (RC) mode. With these clusters going multicore, multiple issues need to be taken into account to design a scalable and high performance collective communication system. One of these issues which is especially significant is the impact of concurrent network transactions over RC transport of InfiniBand. Further, the scalability of RC mode in terms of communication resources is also equally important.

In this paper, we study the utility of shared memory to enhance several aspects relating to performance and resource consumption. Specifically, we study the impact of cutting down of network transactions for important collective operations such as MPI_Barrier, MPI_Allreduce for multicore architectures. For MPI_Alltoall, we evaluated message aggregation techniques and used Analytical Modeling to obtain valuable insights. We also explained the savings obtained in the connection resources for RC transport of InfiniBand. We have evaluated our designs over the quad-core Intel Xeon Clovertown systems. Our results show that an improvement by a factor of three can be achieved for MPI_Barrier and MPI_Allreduce. For MPI_Alltoall, we observe high degradation in performance without using shared memory message aggregation techniques.

For the future we would like to evaluate the shared memory collectives over the UD transport and understand the benefits and potential limitations. We would also like to evaluate the benefits at the application level using the optimizations studied in this paper.

References

[1] M. Banikazemi and D. K. Panda. Can Scatter Communication Take Advantage of Multidestination Message Passing? In *Int'l Symposium on High Performance Computing (HiPC '00)*, 2000.

[2] 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.

[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] 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.

[5] InfiniBand Trade Association. InfiniBand Architecture Specification, Release 1.1. http://www.infinibandta.org, November 2002.

[6] 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 *EuroPVM/MPI*, Oct. 2003.

[7] M. Koop, T. Jones, and D. K. Panda. Reducing Connection Memory Requirements of MPI for InfiniBand Clusters: A Message Coalescing Approach. In *Int'l Symposium on Cluster Computing and the Grid (CC-Grid)*, 2007.

[8] J. Liu, J. Wu, S. P. Kinis, D. Buntinas, W. Yu, B. Chandrasekaran, R. Noronha, P. Wyckoff, and D. K. Panda. MPI over InfiniBand: Early Experiences. Technical Report, OSU-CISRC-10/02-TR25, Computer and Information Science, the Ohio State University, January 2003.

[9] llnl. IBM SP. http://www.llnl.gov/computing/tutorials/ibmsp/.

[10] A. R. Mamidala, S. Narravula, A. Vishnu, G. Santhanaraman, and D. K. Panda. Using Connection-Oriented and Connection-Less Transport on Performance and Scalability of Collective and One-sided operations: Trade-offs and Impact. In *International Symposium on Principles and Practice of Parallel Programming (PPoPP 2007)*, 2007.

[11] A. R. Mamidala, A. Vishnu, and D. K. Panda. Efficient shared memory and rdma based design for mpi-allgather over infiniband. In *EuroPVM/MPI*, 2006.

[12] NASA. NAS Parallel Benchmarks. http://www.nas.nasa.gov/Software/NPB/.

[13] A. R.Mamidala, J. Liu, and D. K. panda. Efficient Barrier and Allreduce InfiniBand Clusters using Hardware Multicast and Adaptive Algorithms . In *Proceedings of Cluster Computing*, 2004.

[14] S. Sistare, R. vandeVaart, and E. Loh. Optimization of MPI collectives on clusters of large-scale SMP's. In *Proceedings of the 1999 ACM/IEEE conference on Supercomputing*, 1999.

[15] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra. *MPI–The Complete Reference. Volume 1 - The MPI-1 Core, 2nd edition.* The MIT Press, 1998.

[16] S. Sur, U. Bondhugula, A. Mamidala, H.-W. Jin, and D. K. Panda. High performance RDMA based All-to-All Broadcast for InfiniBand Clusters. In *(HiPC)*, 2005.

[17] S. Sur, L. Chai, H.-W. Jin, and D. K. Panda. Shared Receive Queue Based Scalable MPI Design for InfiniBand Clusters. In *International Parallel and Distributed Processing Symposium*, April 2006.

[18] 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.

[19] S. Sur, M. Koop, and D. K. Panda. High-Performance and Scalable MPI over InfiniBand with Reduced Memory Usage: An In-Depth Performance Analysis. In *SuperComputing (SC 06)*, 2006.

[20] R. Thakur and W. Gropp. Improving the Performance of Collective Operations in MPICH. In *Euro PVM/MPI*, 2003.

[21] 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.

[22] TOP 500 Rankings. Sun Starfire. http://www.top500.org/orsc/1999/sun.html?q=ORSC/1999/sun.html.

[23] M.-S. Wu, R. A. Kendall, and K. Wright. Optimizing collective communications on smp clusters. In *ICPP 2005*, 2005.