Efficient SMP-Aware MPI-Level Broadcast over InfiniBand's Hardware Multicast[*]

Amith R. Mamidala   Lei Chai   Hyun-Wook Jin   Dhabaleswar K. Panda

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

Abstract

Most of the high-end computing clusters found today feature multi-way SMP nodes interconnected by an ultra-low latency and high bandwidth network. InfiniBand is emerging as a high-speed network for such systems. InfiniBand provides a scalable and efficient hardware multicast primitive to efficiently implement many MPI collective operations. However, employing hardware multicast as the communication method may not perform well in all cases. This is true especially when more than one process is running per node. In this context, shared memory channel becomes the desired communication medium within the node as it delivers latencies which are of an order of magnitude lower than the inter-node message latencies. Thus, to deliver optimal collective performance, coupling hardware multicast with shared memory channel becomes necessary. In this paper we propose mechanisms to address this issue. On a 16-node 2-way SMP cluster, the Leader-based scheme proposed in this paper improves the performance of the MPI_Bcast operation by a factor of as much as 2.3 and 1.8 when compared to the point-to-point and original solution employing only hardware multicast. We have also evaluated our designs on NUMA based system and obtained a performance improvement of 1.7 using our designs on 2-node 4-way system. We also propose a Dynamic Attach Policy as an enhancement to this scheme to mitigate the impact of process skew on the performance of the collective operation.

[*]This research is supported in part by DOE grant #DE-FC02-01ER25506, NSF Grants #CNS-0403342 and #CNS-0509452; grants from Intel, Mellanox, Sun MicroSystems, Cisco Systems, and Linux Networks; and equipment donations from Intel, Mellanox, AMD, Apple, IBM, Microway, PathScale, SilverStorm and Sun MircroSystems.

1. Introduction

Clusters built from commodity PCs have become a popular choice to design high-end computing systems owing to their high performance-to-price ratios. 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.

Message Passing Interface (MPI) [10] 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 [8] employ these collective operations such as MPI_Bcast, MPI_Barrier, etc. To achieve optimal performance, these primitives need to be implemented efficiently by choosing the fastest communication methods offered by the underlying system and the cluster interconnect.

Most of the modern network interconnects provide advanced features which can be utilized for efficient inter-node communication. Recently, InfiniBand has emerged as one of the leaders in the high performance networking domain [3]. A notable feature of InfiniBand is that it provides hardware multicast. By using this feature, a message can be sent to several nodes in an efficient and scalable manner. We have shown the benefits of utilizing this feature for designing collectives such as MPI_Bcast, MPI_Allreduce and MPI_Barrier [5] [4] [9]. Though the proposed solutions scale well for small SMP sizes, they are not beneficial for higher multi-way SMPs. This is because of the extra I/O bus overhead incurred for each multicast packet replicated at the NIC. Though the I/O bus speeds have improved, shared memory channel delivers an order of magnitude lower intra-node message latency. Thus, an optimal choice of communication method for such collectives would be using hardware multicast for distributing the messages across the nodes and utilizing shared memory for distribution within the node.

An important merit of hardware multicast based solutions is the tolerance to process skew. Since the forwarding of the packets is done entirely by the hardware, the processes are decoupled from one another. This assumption is broken while utilizing shared memory channel because processes which are not receiving hardware multicast packets depend on other processes for message forwarding.

In this paper, we aim to answer the following questions:

- *What is the best possible method of coupling hardware multicast with shared memory? Can the existing shared memory solutions apply directly or are new designs required?*
- *What additional mechanisms are needed to break the dependency on the forwarding of hardware multicast packets within a node?*

We explore different design alternatives to tackle these issues. As a solution to the above questions, we propose a Leader-based approach as a suitable method to design an efficient and scalable SMP-Aware MPI-Level Broadcast. Further, as an enhancement to this approach we come up with a Dynamic Attach policy to mitigate the impact of process skew on the collective's performance. We have implemented our designs and integrated them into MVAPICH [6] which is a popular MPI implementation used by more than 295 organizations in 30 countries. We have evaluated our designs on both bus-based and NUMA-based systems. On a 16-node 2-way SMP cluster, our designs show an improvement of as much as 2.3 and 1.8 when compared to the point-to-point and original solution employing only hardware multicast. On a 2-node 4-way NUMA cluster, we obtain a performance gain by a factor of 1.7 using our designs. On a larger cluster and multi-way SMP system, we expect the performance benefits to be even higher.

The rest of the paper is organized in the following way. In Section 2, we provide an overview of the InfiniBand Architecture. 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 InfiniBand Overview and Hardware Multicast

The InfiniBand Architecture (IBA) [2] 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). Channel Adapters usually have programmable DMA engines with protection features. 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.

One of the notable features provided by the InfiniBand Architecture is hardware supported multicast. It provides the ability to send a single message to a specific *multicast address* and have it delivered to multiple processes which may be on different end nodes. Although the same effect can be achieved by using multiple point-to-point communication operations, hardware multicast provides the following benefits:

- With hardware supported multicast, packets are duplicated by the switches only when necessary. Therefore, network traffic is reduced by eliminating the cases that multiple identical packets travel through the same physical link.
- Since the multicast is handled by hardware, it has very good scalability.

In InfiniBand, before multicast operations can be used, a multicast group which is identified by a multicast address must be created. Creating and joining multicast groups can be achieved through the help of InfiniBand Subnet Manager and Subnet Administrator.

Another important operation performed by the process to receive the hardware multicast packet is the Attach operation for a specified multicast group as indicated in Fig. 1. It is a way of notifying the NIC that the multicast packet has to be delivered to a particular process. The opposite of Attach is the Detach operation which detaches the process from the multicast group.

In InfiniBand, hardware multicast operation is only available under the Unreliable Datagram (UD) transport service. In UD, a connectionless communication model is used. Messages can be dropped or arrive out of order. In our earlier work [5], we have designed schemes to take care of reliable MPI_Bcast on top of IBA hardware multicast. In [7], we have taken care of solutions for supporting multiple multicast groups with IBA hardware multicast. However, all these solutions can not work with SMP architecture and hence the current schemes are not SMP-Aware. This is the focus of our research in this paper.

2.2 Shared Memory Channel in MVAPICH

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 messages are sent eagerly. The sending process copies the message along with other information required for message matching to the shared memory area. The receiving process can then match the tags of the posted receives and accordingly copy over the correct message to its own buffer. For large messages, redezvous protocol is used. The sending and receiving processes first exchange handshaking messages through the shared memory region, and then the data payload is packetized and sent out in a pipelined manner. This approach involves minimal setup overhead for every message exchange and shows better performance than NIC-level message loopback.

3 Why Hardware Multicast is not enough?

As illustrated in Fig. 1, when a multicast packet arrives at the NIC, it has to be forwarded to the processes attached to the multicast group specified in the packet header. This process comprises of three steps. First, the NIC has to look up the queue-pairs of the processes which are attached to the multicast group. It then replicates the packets and in the final step DMAs the data to each process's buffer. This cost increases with the increase in the number of processes attached to the multicast group because these DMA operations are sequentialized.

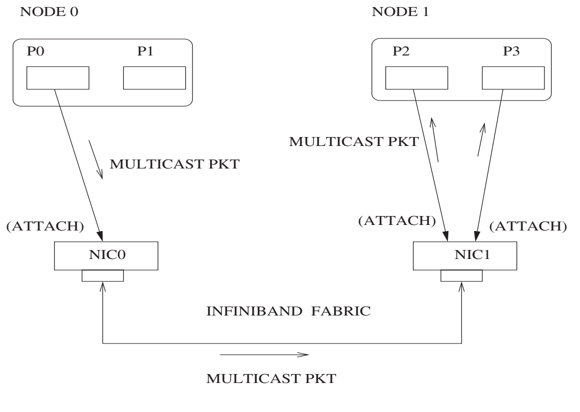

Figure 1. Operational principle of IBA Hardware Multicast

Fig. 2 compares the latency of the MPI_Bcast over hardware multicast for two configurations, 4x2 (meaning two processes per node across four nodes) and 8x1. As shown in the figure, the latency of the former case is significantly higher than the latter. This is because as explained above, in the 4x2 case the DMA of the multicast packets is sequentialized at the NIC resulting in higher latencies. This demonstrates that we need a mechanism to handle multicast based collective efficiently over multi-processor nodes.

Fig. 3 illustrates the difference between the latency of point-to-point inter-node communication vs the latency of intra-node communication via shared memory channel. As indicated in the figure, the latencies of intra-node channels is an order of magnitude smaller than the inter-node latencies. These two observations lead to the following question:

Can we utilize both shared memory channel and Hardware multicast of InfiniBand to design efficient collectives while running more than one process per node?

In this paper we propose an efficient approach to leverage the shared memory channel for intra-node messages and hardware multicast for inter-node messages to implement MPI_Bcast operation.

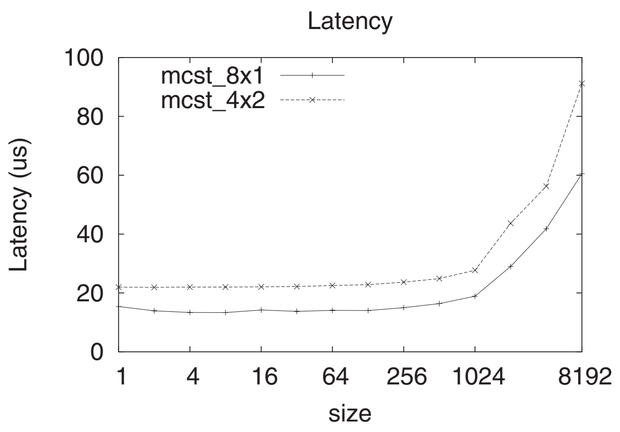

Figure 2. MPI_Bcast Latency with IBA hardware multicast on two system configurations of 8 nodes

4 Design and Implementation

There are two important design alternatives for implementing collectives utilizing both shared memory channel and hardware multicast.

- Direct multicast to shared memory: In this approach the multicast packets can be received directly into the shared memory regions across the nodes. After detecting the multicast message arrival, the local processes can copy the message into its own buffer. Though this approach looks promising, detecting the arrival of

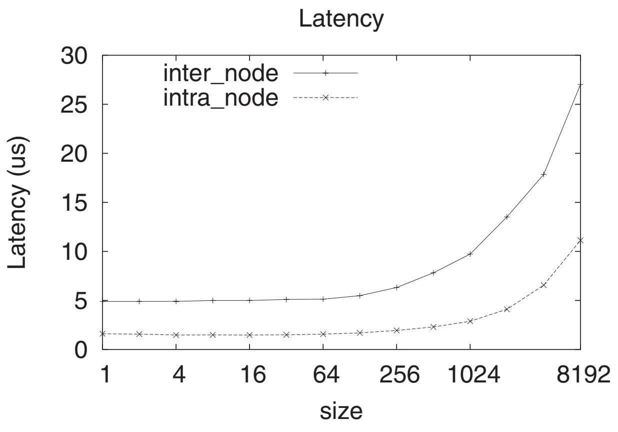

Figure 3. Comparison between inter-node and intra-node Point_to_Point Latency

the multicast packet is a tricky and complex operation. This is because the arrival of the message is notified only to the processes which are attached to the multicast group. As discussed in the earlier sections, this approach does not scale well. Another approach would be to write a separate flag following the message. This solution also does not guarantee correctness as there is no ordering guarantee between the UD-based multicast messages and UD or RC based point-to-point flag messages.

- Leader-based Approach: In this approach a designated process receives hardware multicast messages. It can then distribute the message to the remaining nodes.

We have taken the latter approach in the paper which is described in the following section.

4.1 Leader-based Approach

In this approach as indicated in Fig. 4, the broadcast operation occurs in a hierarchical two-step manner:

1. The Root process posts a multicast message to the multicast group. A set of Leader processes is identified, one Leader per node which receives the multicast packets. These leaders have to be attached to the multicast group at the NIC to receive the packet.

2. The Root after posting the multicast message delivers the message to the participating local processes via shared memory channel. The Leaders on each node do the same after receiving the multicast packet from the Root.

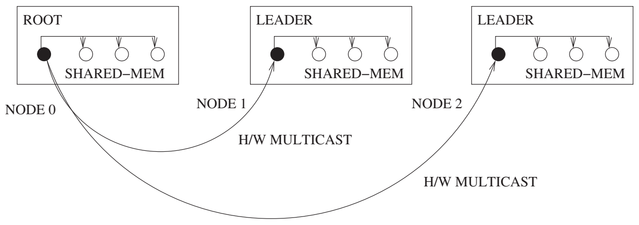

Figure 4. Leader-based design

The leaders are identified using the local IDs which are initialized during the initialization of the shared memory channel. In our implementation, the process with local ID zero is chosen as the Leader which receives the hardware multicast packets. It forwards the packets to the other processes by indexing into the local-to-global rank mapping table to determine the other local processes running on the node. This table is also set up during the initialization phase of the shared memory channel.

It is to be noted that hardware multicast is unreliable in InfiniBand. In [5] we have already proposed Ack based reliability schemes to address the problem. In these schemes, the processes have to send Acks to confirm the receipt of the message. In the approach described here, only the Leaders are required to send the Acks as the other processes receive intra-node messages over shared memory channel.

4.2 Dynamic Attach Policy

The basic design indicated above does not always deliver good performance. This situation occurs when both the hardware multicast packet and a non-Leader process have arrived at the collective call but the Leader process did not arrive. In this case, the non-Leader process has to wait for the Leader to forward the message. This can hamper the performance of the application especially if the chosen Leader always arrives late.

The best possible solution would be for the first process to arrive at the collective to attach itself to the multicast group and become the Leader process. We may need to take care of the race condition of the packet arriving before the attach operation. And, after the collective operation is done, the Leader process detaches from the multicast group. However, the attach and detach overheads measured on current generation systems, are in the order of 100 us and are thus prohibitively expensive for this kind of approach. Hence, we choose an alternative method as described below.

We now describe a Dynamic Attach policy wherein a non-Leader process attaches to the multicast group based on certain conditions. The non-Leader process uses the average wait-time computed across different broadcasts to make a decision. The average wait-time after a certain num-

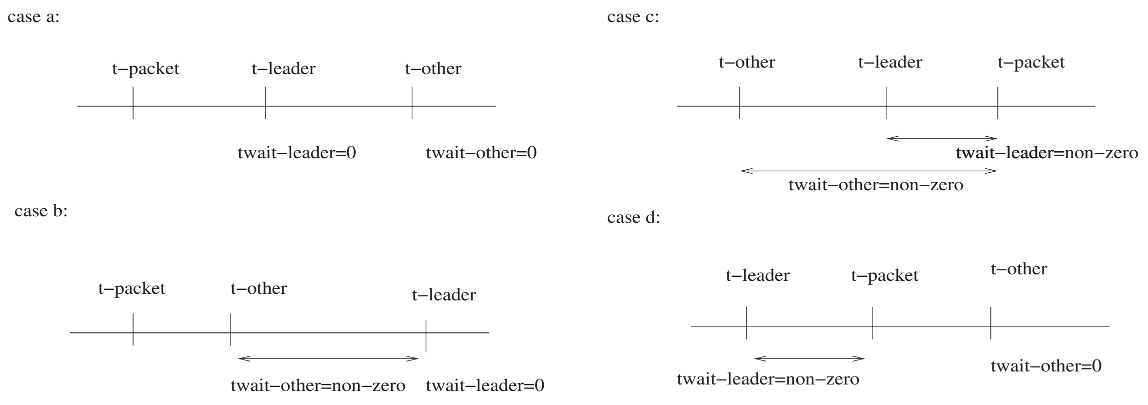

Figure 5. Dynamic Attach Policy

ber of broadcasts is the total wait-time accumulated so far divided by the number of broadcast operations completed. The wait-time can be easily computed by computing the difference between the time when the message is received and the time when the broadcast is invoked.

Fig. 5 illustrates the possible cases of the arrival of the packet, the Leader and the other process which is not the leader. We make an assumption that the wait times are much higher than the intra-node latencies. Under this condition, the wait-times for both the Leader and the other process are zero when the packet is unexpected (i.e. arrives before the receiver) as shown in case a. The wait time for the non-Leader process is zero or less than that of the Leader in case d. The only cases for which it has a wait time greater than that of the Leader are cases b and c.

If we can discount case c, then we can safely say that if the average wait times of the non-Leader process are higher than the Leader process, then case b is the most frequently happening case. In this case, the non-Leader process can attach itself to the multicast group rather than waiting for the Leader process to forward the messages. However, if case c happens more frequently then this assumption would not hold true as in this case both the leader and the other process are waiting for the packet to arrive.

We can overcome the above problem by making the non-Leader process compute the average only when the packet is expected to the Leader, as in case b. This can be easily implemented by the Leader setting a flag in the shared memory buffer. The average wait time of the Leader process is also stored in a shared memory region. The non-Leader process computes the wait time by starting the timer when it enters the collective and stopping it after receiving the packet. It recomputes the average wait time when the flag is set. It chooses to attach to the multicast group when the difference between its average wait time and that of the Leader's wait time crosses a threshold value. Conversely, if this difference decreases below the threshold, these non-Leader processes detach from the multicast group. To avoid repeated attach and detach overhead, the non-Leader processes detach in a lazy manner, i.e. they wait for some number of iterations before detaching. Also, the proposed policy comes into effect only after the number of broadcasts executed cross a threshold. This is to decrease error margins which can occur if the average wait time is based on too few broadcasts.

5 Performance Evaluation

In this section we compare the performance of the new scheme proposed in the paper with the already existing approaches. The comparison is made by running the *broadcast latency* micro-benchmark for all the schemes across different message sizes. To show the benefits of the dynamic attach policy, we have modified the *broadcast latency* micro-benchmark to add skew within the node.

All the different schemes considered and their abbreviations are as follows:

- smp_mcst: The new SMP-Aware solution proposed in the paper.
- nosmp_mcst: The original solution employing hardware multicast but no shared memory channel as proposed in [5]
- ori_bcst: The point-to-point implementation utilizing network for inter-node communication and shared memory for intra-node communication

To compare the benefits of the dynamic attach policy we have run the modified *broadcast latency* program explained above with and without the dynamic attach support.

Another important consideration while running the multiple processes per node is how they are distributed across the nodes in a cluster. In the *cyclic* distribution consecutive processes are assigned to different processors while in *block* distribution they are not assigned to different processors until the node has reached its capacity with repect to the number of processors.

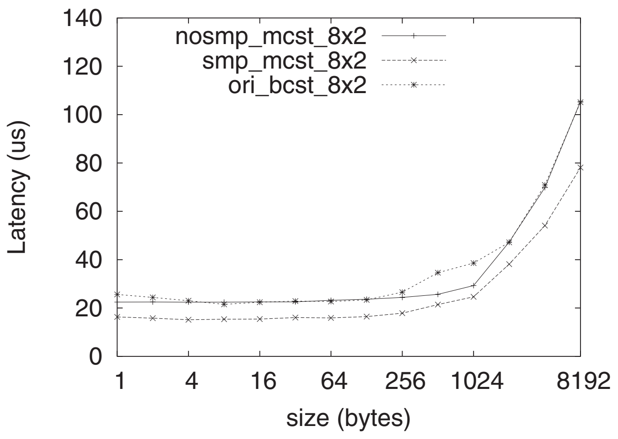

Figure 6. Broadcast Latency, Cluster A: 8x2

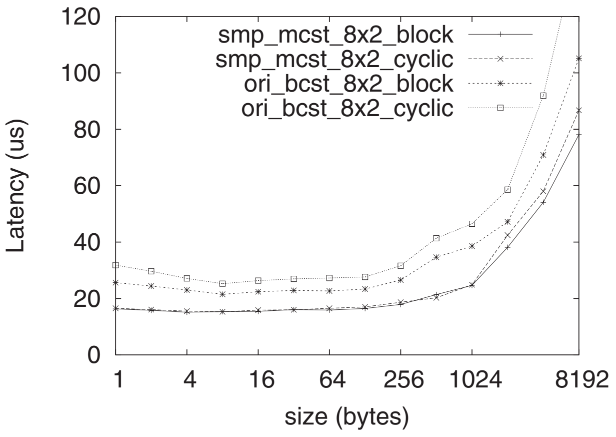

Figure 9. Broadcast Latency, Cluster A: 8x2

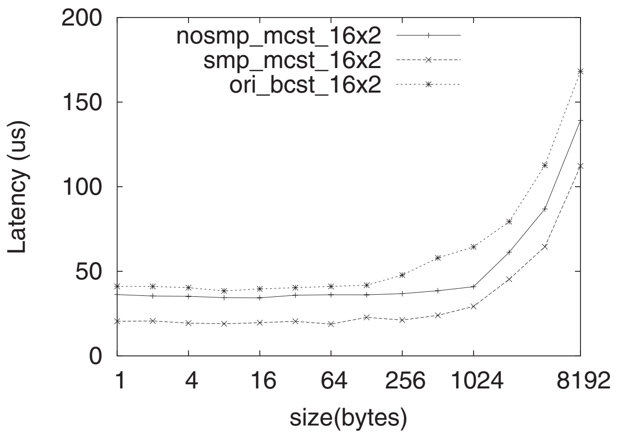

Figure 7. Broadcast Latency, Cluster A: 16x2

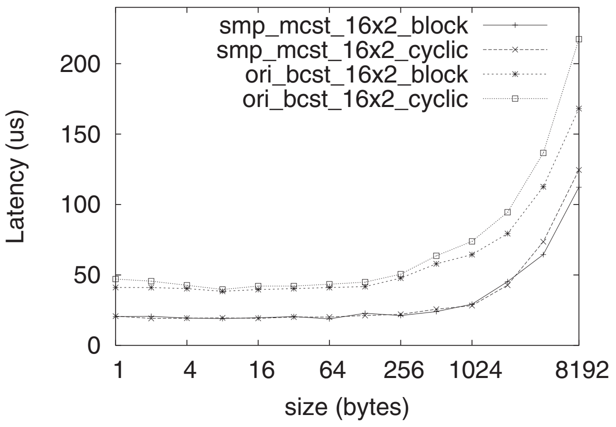

Figure 10. Broadcast Latency, Cluster A: 16x2

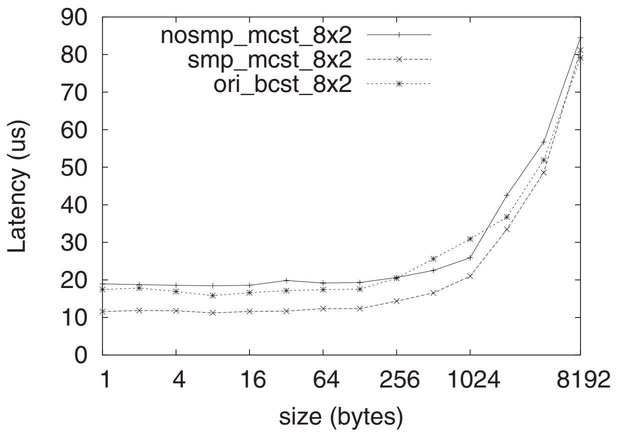

Figure 8. Broadcast Latency, Cluster B: 8x2

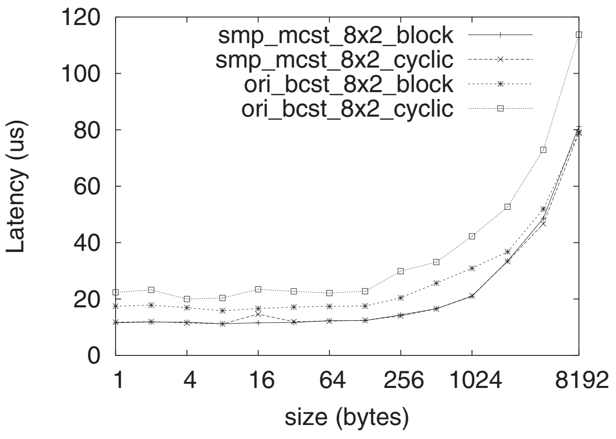

Figure 11. Broadcast Latency, Cluster B: 8x2

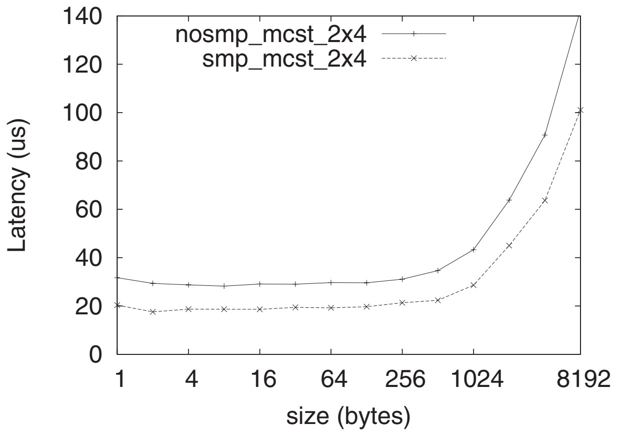

Figure 12. Broadcast Latency, Cluster C: 2x4

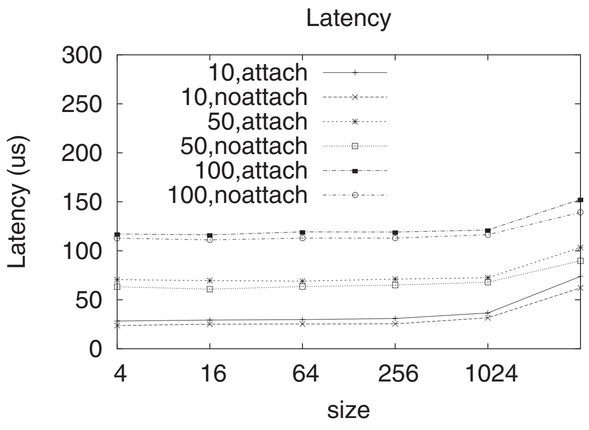

Figure 13. Impact of Dynamic Attach on Broadcast Latency when Leaders arrives late

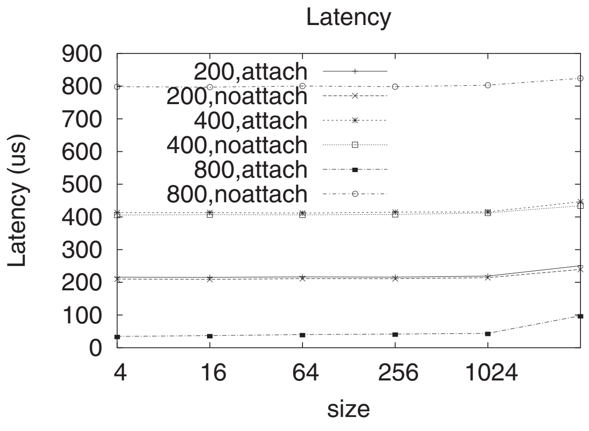

Figure 14. Impact of Dynamic Attach on Broadcast Latency when Leaders arrives late

5.1 Experimental Testbed

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

Cluster A: This cluster consists of 16 SuperMicro SUPER X5DL8-GG nodes with ServerWorks GC LE chipsets. Eight of these are Intel Xeon 3.0 GHz processors and the other are Intel Xeon 2.4 Ghz with , 512 KB L2 cache, and PCI-X 64-bit 133 MHz bus. We have used InfiniHost MT23108 DualPort 4x HCAs from Mellanox.

Cluster B: This cluster consists of 8 dual Intel Xeon 3.2GHz EM64T systems. Each node is equipped with 512MB of DDR memory and PCI-Express Interface. These nodes have MT25128 Mellanox HCAs with firmware version 5.1.0. The nodes are connected by an 8-port Mellanox InfiniBand switch.

Cluster C: This cluster consisting of 2 quad Opteron systems. Each of the nodes have MT23108 DualPort 4x HCAs from Mellanox. All nodes are connected to a single Mellanox InfiniScale 24 port switch MTS 2400, which supports all 24 ports running at full 4x speed.

5.2 Broadcast Latency

Broadcast latency is the time taken for a broadcast message to reach every receiver. The test consists of a loop, in which an MPI_Bcast is issued from a root node and the receivers take turns to send back an acknowledgment using MPI_Send. The broadcast latency is derived from time to finish each iteration and the MPI point-to-point latency.

Figures 6 and 7 show the latency of the broadcast operation for 16 and 32 processes, respectively. The number of processes per node is equal to two and we use a block distribution to scatter the processes. The cyclic distribution is considered separately. As indicated in the figure, the performance of nosmp_mcst is comparable to ori_bcst for 16 and 32 processes. However, in both cases the smp_mcst performs well delivering performance improvement by a factor of 2.18 and 1.8 when compared to ori_bcst and nosmp_mcst, respectively for 32 processes. Similar trends are also observed on Cluster B, Fig. 8.

Figures 9 and 10, we compare the *broadcast latency* for the cyclic and block distributions. The new design proposed in this paper does equally well both with cyclic and block distributions. However, the ori_bcst scheme performs better in the block distribution vs cyclic distribution. This is because though the messages within a node are transferred over shared memory, the MPI implementation has no global knowledge of local and remote processes. As a result, in the cyclic distribution case the intra-node messages are first delivered and then the inter-node messages. This weakens the hierarchical design which is essential for maximum overlap between the inter- and intra-node communication. The same results are also obtained on Cluster B, as indicated in Fig. 11. Further, in Fig. 12, the benefits of shared memory are cleary visible for a 4-way NUMA system. Shared memory design improves the latency by a factor of as much as 1.7.

In the final tests, we have added skew to the *broadcast latency* test by delaying the leaders by variable amounts. We compare the new design proposed in the paper together with the Dynamic Attach enhancement. The threshold selected was equal to 500 μs which is the result of multiplying processors per node and the attach latency which was measured to be around 250 μs. As shown in Figures 13 and 14, the latency of the operation increases as the delay is being added to the leader. The legend indicates the delay added followed by the scheme selected. As the delay is increased, the original scheme with no dynamic attach keeps incurring the cost but where as the scheme with dynamic attach drops down after crossing the threshold. This occurs for the delay value of 800 μs for the leader.

6 Related Work

Utilizing shared memory for implementing collective communication has been a well studied problem in the past. In [11], 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 operation. Our approach differs from theirs as the inter-node communication method employed by the authors is based on point-to-point RDMA operations unlike our current approach which is based on one-to-many hardware multicast. In [1], the authors implement collective operations over Sun systems. In [12], the authors improve the performance of send and recv operations over shared memory and also apply the techniques for group data movement.

7 Conclusions and Future Work

In this paper we proposed a Leader-based mechanism to couple InfiniBand's hardware multicast communication with the shared memory channel to deliver optimal performance to the MPI collectives. This is especially true for the modern systems which feature multi-way SMPs allowing more than one process to run on a single node. Our results show that the scheme proposed in the paper delivers a performance improvement by a factor of as much as 2.3 and 1.8 when compared to the point-to-point and original solution employing only hardware multicast. Also, on a 4-way NUMA system we observed a performance gain of 1.7 with our designs. We also propose a Dynamic Attach policy to alleviate the performance bottlenecks caused due to process skew. As a future work, we plan to extend the Leader-based approach to other collectives as well such as Allreduce, Allgather, etc. We also plan to extend the Dynamic Attach policy to implement a dynamic Leadership change protocol. As described in the paper, the high overhead of attach and detach prohibit this kind of approach to be taken in the paper. We expect this overhead to reduce with the upcoming versions of InfiniBand drivers. We also plan to evaluate the mechanisms proposed in the paper on multi-way(4-way, 8-way) SMP clusters and by doing application level studies.

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

[3] InfiniBand Trade Association. InfiniBand Architecture Specification, Release 1.1. http://www.infinibandta.org, October 2004.

[4] 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.

[5] J. Liu, A. R.Mamidala, and D. K. panda. Fast and Scalable MPI-Level Broadcast using InfiniBand's Hardware Multicast Support. In *Proceedings of IPDPS*, 2004.

[6] 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.

[7] A. R. Mamidala, H.-W. Jin, and D. K. Panda. Efficient hardware multicast group management for multiple mpi communicators over infiniband. In *EuroPVM/MPI*, 2005.

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

[9] 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.

[10] 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.

[11] 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.

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