Designing Multi-Leader-Based Allgather Algorithms for Multi-Core Clusters *

Krishna Kandalla, Hari Subramoni, Gopal Santhanaraman, Matthew Koop and Dhabaleswar K. Panda

Department of Computer Science and Engineering, The Ohio State University
{*kandalla, subramon, santhana, koop, panda*}*@cse.ohio-state.edu*

Abstract

The increasing demand for computational cycles is being met by the use of multi-core processors. Having large number of cores per node necessitates multi-core aware designs to extract the best performance. The Message Passing Interface (MPI) is the dominant parallel programming model on modern high performance computing clusters. The MPI collective operations take a significant portion of the communication time for an application. The existing optimizations for collectives exploit shared memory for intra-node communication to improve performance. However, it still would not scale well as the number of cores per node increase. In this work, we propose a novel and scalable multi-leader-based hierarchical Allgather design. This design allows better cache sharing for Non-Uniform Memory Access (NUMA) machines and makes better use of the network speed available with high performance interconnects such as InfiniBand. The new multi-leader-based scheme achieves a performance improvement of up to 58% for small messages and 70% for medium sized messages.

1. Introduction

The use of high performance computing has been growing continuously over the last decade as the need for additional computational cycles continues to exceed availability. Though modern high performance interconnects such as InfiniBand [**?**] address the growing bandwidth requirements of current day high performance computing clusters, much of the additional computing power for these systems has been achieved through the use of multi-core processors. These additional cores per compute node add more choices into software design for high performance computing.

The Message Passing Interface (MPI) [**?**] is the dominant parallel programming model on current generation high performance computing clusters. Given the role of the MPI library as the communication substrate for applications, the library must be designed to be multi-core aware to achieve the highest performance. The MPI standard defines two main types of communication: point-to-point and collective. The collective operations involve multiple processes in the job. These operations commonly take a large portion of the communication time for an application. If these collective operations can be designed to be multi-core aware, the overall performance of an application can improve [**?**].

Currently, optimized collectives for multi-core machines [**?**], [**?**] use shared memory on each node to aggregate messages and have a single leader per node responsible for inter-node exchanges. This increases the degree of communication that takes place on the node and reduces network traffic. While our studies have shown that this can increase performance, it still ignores the multi-core aspect of the machines and will not scale as the number of cores per node increases.

In this paper we propose a novel *multi-leader-based* approach for collective communication. We consider MPI_Allgather, an important collective used in applications like matrix multiplication kernels [**?**] and video compression [**?**], and show how it can be transformed into a multi-leader-based approach. We design the multi-leader-based hierarchical model of communication by considering both point-to-point-based as well as shared-memory-based schemes. We incorporate our design into MVAPICH2 [**?**], a popular MPI library over InfiniBand and evaluate them on high performance computing clusters. Our evaluations show a latency improvement of up to 70% for a range of message sizes.

The rest of the paper is organized as follows. Section 2 gives the background needed for this paper. Existing designs for MPI_Allgather and their limitations are presented in Section 3. In Section 4, we present our new multi-leader-based collective design for MPI_Allgather. Section 5 details the performance evaluation results of the various design options against the existing single-leader-based designs and the conventional MPI_Allgather algorithms. Research work related to ours is presented in Section 6. Our conclusions and thoughts for future work are presented in Section 7.

* *This research is supported in part by U.S. Department of Energy grants #DE-FC02-06ER25749 and #DE-FC02-06ER25755; NSF grants #CNS-0403342, #CCF-0702675 and #CCF-0833169; grant from Wright Center for Innovation #WCI04-010-OSU-0; grants from Mellanox, Intel, Cisco, QLogic and Sun Microsystems; Equipment donations from Intel, Mellanox, AMD, Advanced Clustering, Appro, QLogic and Sun Microsystems.*

978-1-4244-3750-4/09/$25.00 ©2009 IEEE

2. Background

In this section we briefly describe the required background for our work.

Multi-Core Architecture: In multi-core processors, we have two or more independent cores combined into a single die. The cores may share a single coherent cache as in the case of Intel machines or have separate caches like the AMD multi-core processors [**?**]. In this work, we use the AMD multi-core platform for evaluating our proposed designs. AMD multi-core Opterons [**?**] are based on NUMA [**?**] architecture, with sockets providing the necessary electrical interface between the motherboard and the processing cores. Each of the sockets share independent memory units. The latest Barcelona systems have four cores per socket. All of the cores have independent L2 caches. Point-to-point HyperTransport [**?**] links provide the required bandwidth scalability between the cores. Further, these links are connected by a 2-D mesh topology providing for scalable and less congestion-prone on-chip interconnection network. Also, the future Intel designs are expected to be NUMA based to allow for increased memory bandwidth. These designs will use the Intel Quick Path Interconnect (QPI) [**?**] as the interconnect between the sockets.

InfiniBand: The InfiniBand Architecture (IBA) [**?**] 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 the Channel Adapters (CA). In recent years, several large scale InfiniBand clusters have been deployed with many featuring in Top 500 list of supercomputers.

MPI_Allgather Collective: The Allgather collective operation is also known as the Alltoall-Broadcast and it involves each process broadcasting a vector data to every other process in the group. If there are N processes involved, the MPI_Allgather operation can be visualized as N concurrent broadcast operations with each process acting as the root. This kind of operation is heavily used in matrix multiplication kernels.

MVAPICH2: MVAPICH2 is a high performance implementation of MPI-2 over InfiniBand. MVAPICH2 is available as an open source distribution and is currently being used by more than 840 organizations worldwide including several several high end computing platforms including the TACC Ranger system [**?**] which is currently ranked the sixth fastest supercomputer.

3. Existing Approaches for MPI_Allgather

In this section, we describe the existing schemes for MPI_Allgather.

3.1. Conventional MPI_Allgather Designs

The most popular algorithms for MPI_Allgather are the recursive doubling scheme for short messages and the ring algorithm for large messages [**?**]. The recursive doubling algorithm is based on the hypercube scheme and requires $log(p)$ steps to complete, where p is the number of processes involved. In each step, the distance between the peers exchanging data increases by a factor proportional to power of 2 and the size of messages exchanged also doubles. If we consider ts to be the start-up cost for communication between two processes and tw to be the time required to exchange a word of data between the two processes in a single ported model with bi-directional links, the cost of communication for the recursive doubling scheme is $(ts * logp + tw * (p-1) * m)$. The ring algorithm requires p steps of communication to complete, with a process exchanging data with its left and right neighbor in every iteration with the amount of data exchanged being the same across all the iterations. The cost of communication for the ring scheme is known to be $(ts + tw * m) * (p-1)$. Since the recursive doubling algorithm requires lesser number of communication steps, the start-up costs are lower and it is optimal for small and medium messages. However, it has been observed that the ring scheme performs better for larger messages and this has been attributed to its near-neighbor communication pattern. These algorithms were designed and optimized for single process per node systems over Ethernet networks. All of these algorithms are currently employed in MPICH2 [**?**].

In MVAPICH2, all the collective algorithms including the MPI_Allgather operations described above have default implementations over point-to-point operations. These are derived from MPICH2 implementation.

3.2. Single-Leader-Based Collective Design

Single-leader-based hierarchical approaches have been proposed for multi-core clusters [**?**], [**?**] to address this issue and optimize the performance of collectives. In [**?**], [**?**], authors propose similar schemes for Symmetric Multi-Processor (SMP) based systems. In [**?**], authors propose approaches that choose one process per node as a $leader$ and the other processes that reside on the same node are referred to as the $children$. One simple way to select the leader process would be to choose the lowest ranking process on each node. The entire algorithm could be broken up into three phases:

1) Aggregation of messages at the leader processes
2) Data exchange between leaders
3) Distribution of data from the leaders to children

Phases (1) and (3) are necessarily intra-node operations and phase (2) will entail inter-node exchanges.

The inter-node exchanges boil down to one process in one node exchanging data with only one process in other

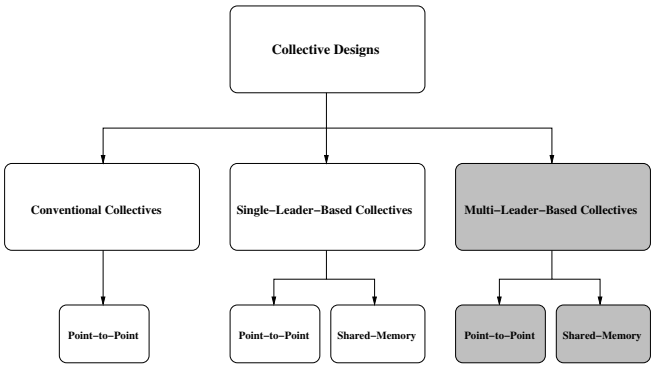

Figure 1. Collective Design Alternatives (White - Existing, Gray - Proposed)

nodes, which is identical to the traditional clusters with single process per node. As explained above, for such cases, recursive doubling scheme performs better for smaller messages and the ring scheme is more optimal for larger messages. For intra-node exchanges there are potentially two approaches: (i) point-to-point-based and (ii) shared-memory-based.

In point-to-point-based approaches, the data transfers in phase (1) and phase (3) are implemented on top of point-to-point MPI_Send and MPI_Recv operations. In MVAPICH2, these calls are optimized to use the SMP channel for intra-node exchanges. However, despite using the non-blocking versions of these calls, the entire communication operation is still serialized. In the shared-memory-based approach, the leader sets up a shared memory buffer and all the other processes read/write their contribution into this shared memory space concurrently and could lead to better performance.

3.3. Limitations of Existing Schemes

The conventional schemes have severe limitations when used on modern multi-core systems. They do not take into account the hierarchy introduced by the advent of multi-core architectures. They also fail to take advantage of shared memory for communication across cores residing on the same node. In the worst case scenario, where a cyclic distribution [**?**] is used, the ring communication could lead to very high network traffic since every data-transfer could go across the network. This could lead to severe performance degradation. This condition can be partially alleviated using a block distribution of processes [**?**].

The issues related to the conventional schemes have been addressed in the single-leader-based hierarchical approaches. These studies have shown the potential for higher performance by using the hierarchical approaches. However, with the number of cores per node constantly increasing, there is a need to investigate the single-leader-based hierarchical schemes further. While its obvious that intra-node communication can be improved by using shared memory buffers, it is also important to consider the differences in communication characteristics between inter-node and intra-node exchanges. In hierarchical schemes, due to aggregation and distribution of the individual messages, the number of exchanges that occur between nodes is lower than the conventional schemes and help in lowering the communication start-up costs. However, the size of each message sent over the network is higher, which increases the bandwidth component of the communication cost. It is necessary to design schemes that minimize or eliminate network contention as it could severely degrade the performance of hierarchical schemes. It is important to note that these schemes entail making multiple copies of messages and this could be play a role in limiting the performance as the message size increases. With the number of cores per node constantly increasing, it is important to consider the contention that is bound to occur within a compute node and it is necessary to design schemes to mitigate these issues. It is necessary to be cautious when working with schemes that use shared caches due to issues like cache thrashing and page alignment as they can severely degrade the performance of schemes that try to exploit shared memory to optimize the intra-node communication steps [**?**]. The existing hierarchical schemes have addressed the issues related to contention free inter-node exchange and careful cache-aware usage of the shared memory buffers [**?**]. However, they still do not address the issues related to optimizing as intra-node exchanges. As the number of cores per node increases, the contention within each node leads to lower scalability and poor performance of the single-leader-based approach and it is necessary to design schemes to be address this issue.

4. Design of Multi-Leader-Based MPI_Allgather Algorithms

In this section we explain the design and motivation for the multi-leader-based MPI_Allgather algorithms for multi-core architectures.

Earlier works [**?**], [**?**], [**?**] have shown that leader-based schemes could increase performance of the MPI_Allgather collective. From our analysis, however, we found that this method can have very high contention as the number of

functional cores per node increases. As noted in Section 3, the hierarchical schemes can be implemented using either point-to-point calls or shared memory. In the point-to-point-based design we suspect the leader being a bottleneck in processing messages. In the shared-memory-based design, we suspect that the memory contention between processes became significant.

As a result, we propose a *multi-leader-based* design to reduce the contention on memory and at the leader process. Figure 1 shows the overall framework that is being designed. The un-shaded sections of the figure are the conventional designs and the existing single-leader-based hierarchical designs. The shaded sections of the figure correspond to the new multi-leader-based hierarchical designs that we are proposing.

4.1. Number of Leaders

The first major choice when designing a multi-leader-based collective is the number of leaders that each node should have. Existing designs have used only a single-leader-based approach. In Section 3, we described the issues faced with such designs. Increasing the number of leaders per node is a way to alleviate such problems. In particular, as more architectures become NUMA based, sharing buffers across sockets in the system can lead to lower than expected performance. We propose using one leader per socket to alleviate these problems. Since the time required to complete the inter-leader exchange phase is dependent on the number of leaders across all the nodes, it is not optimal to further increase the number of leaders per node.

4.2. Leader Selection

Once the number of leaders has been decided, the next design choice is to select the leaders. Memory contention is the key performance issue being addressed in the multi-leader-based approach. We opt to place leaders such that there is at most one per socket. In this manner, the memory for each subgroup of tasks on a node can reside connected to the same socket as they are executing on. Please note that the selection of a core to act as a leader is not a dedicated arrangement. This core also performs the standard computation and communication phases of the application that is calling MPI_Allgather. During a call to MPI_Allgather, this process does a few dedicated operations to improve the performance.

4.3. Inter-Process Communication

There are multiple design alternatives for the data transfer within each subgroup and between the leader processes.

4.3.1. Subgroup Data Aggregation. During the aggregation step, a child process is involved in data exchange with the group of processes which share the same leader or only with its leader. As design options, we propose the following two schemes to aggregate messages at the leaders.

- *Point-to-Point*: We explore various ways to design the intra-node exchange phase by using point-to-point calls. As design options, we consider using the recursive doubling and the ring algorithms among the processes that share the same leader process. We also carry out experiments that use MPI_Gather for aggregating messages.
- *Shared Memory*: Each leader allocates a shared memory region and all the children in the subgroup directly write their contributions to the buffer. In the current design, we have considered the shared-memory-based approach only for small and medium sized messages. Any attempt to design shared memory collectives for larger messages will have a significant impact on the memory foot-print of the MPI implementation.

4.3.2. Inter-Leader Data Exchange. In this phase, all the leaders exchange data bundles which comprise of their own contributions and the messages received from their children. As design options, we propose using recursive doubling for small messages and ring algorithm for larger messages.

4.3.3. Subgroup Data Distribution. In this phase, the leaders distribute the data they received from other leaders to their children. If we have used ring or recursive doubling approaches for the aggregation phase, all the children participate in the exchange. So, at the end of the step, the children have the contributions of their leader process and the other non-leaders that belong to the same sub-group. So, we can efficiently implement the distribution phase by ensuring that the leader only sends the new data that it received from the other leader processes. On the contrary, if we have used the gather method during the aggregation step, we need to have the leader processes broadcast all of the data that it has at the end of the leader-exchange phase. In the shared-memory-based design, the leaders write the data they have received at the end of the inter-leader exchange phase into the shared memory buffer and each child can simply read from the buffer.

In Figure 2, we show the mapping of the processes within the nodes. We assign one process per socket as the leader and we show the aggregation and the distribution phases. In the figure, the leader processes have been shaded. We also have an ellipse that encapsulates all the leader processes, to symbolize the inter-leader exchange phase. In the figure, the leader process is shown to be distinct from the other processes in the same socket. This is done purely to highlight the difference in the roles between leaders and non-leaders from the perspective of the proposed designs for MPI_Allgather. As explained in Section 4.2, the leader process also performs standard communication and computation phases of the application.

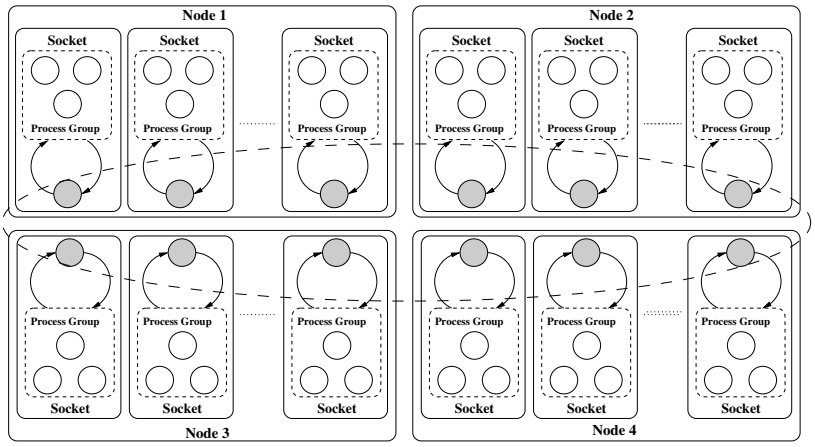

Figure 2. Process Communication Pattern

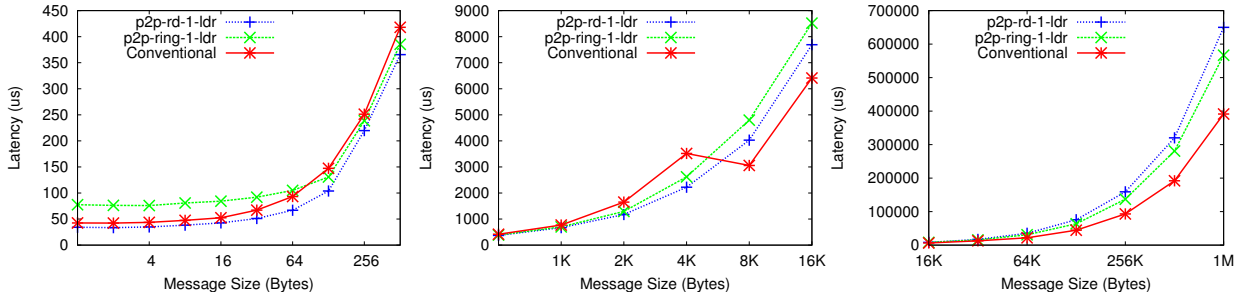

Figure 3. Performance of Single-Leader-Based Point-to-Point Hierarchical Model for: (a) Small messages, (b) Medium messages and, (c) Large messages

5. Performance Evaluation

In this section we evaluate the performance of our proposed designs.

5.1. Experimental Testbed

Each node of our testbed has 16 AMD Opteron 1.95 Ghz processors with 512 KB L2 cache. We ran our experiments on 4 such nodes with 64 processes. Each node also has 16 Gigabyte memory and PCI-Express bus. They are equipped with MT25418 HCAs with PCI-Ex interfaces. A 24-port Mellanox switch is used to connect all the nodes. The operating system used was RedHat Enterprise Linux Server 5. We have used a simple benchmarking tool to evaluate the performance of the various schemes proposed. The benchmark computes the average time required for a call to MPI_Allgather to complete for a set of message sizes. In the following set of graphs, we have used a set of acronyms in the legends to conveniently highlight the various schemes. We use the label $Conventional$ to refer to the current schemes used in MVAPICH2. We use $p2p$ to refer to the schemes that use point-to-point calls for the intra-node exchange phase and $shmem$ for the schemes that use a shared memory buffer. To indicate the number of leaders used for a given scheme, we use ldr which is preceded by a number.

5.2. Evaluation of Single-Leader-Based scheme

We first compare the base version of MVAPICH2 with the different single-leader-based point-to-point hierarchical model that we described in Section 3. We will henceforth refer to this base version as the $Conventional$ scheme, as it is not multi-core aware. We had proposed a set of schemes for the aggregation and the distribution phases for the hierarchical approach. However, on running the experiments, we could infer that those schemes had little variations between them. For the purpose of evaluation, we consider the best of those schemes along with the ring and recursive doubling schemes for inter-leader exchanges. In Figures 3 (a), (b) and (c), we present the comparison of the single-leader-based point-to-point schemes and the existing schemes. For small and medium message sizes, we observe that there is potential for good performance with hierarchical approaches. Also, for these message sizes, it is more beneficial to use recursive doubling for inter-leader communication than the ring scheme. However, beyond a point, the existing ring scheme starts out-performing the single-leader-based point-to-point hierarchical model. We believe that this could be because of the bottleneck at the leader processes at each

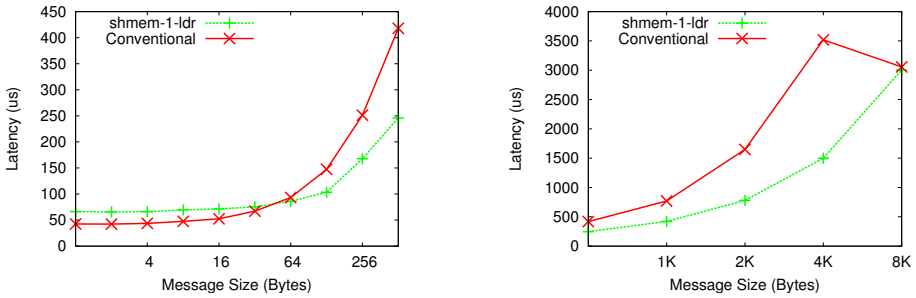

Figure 4. Performance of Single-Leader-Based Shared Memory Hierarchical Model for: (a) Small messages and, (b) Medium messages

node.

In Figures 4 (a) and (b), we show the performance of the single-leader-based shared memory hierarchical model. Since we are considering only small and medium sized messages for the shared memory scheme, we have used recursive doubling for inter-leader exchanges. We can see that there are considerable advantages of using the shared memory approach for medium sized messages. However, for very small messages the existing recursive doubling scheme performs better. As explained earlier, this could be attributed to the memory contention within each node.

By analyzing the performance patterns in these figures, we could see some benefits of using the single-leader-based hierarchical approach. But, we have also identified the issues that were causing this approach to perform sub-optimally under certain conditions in Section 3.3.

5.3. Evaluation of Multi-Leader-Based Scheme

In the following set of graphs, we present the performance of the multi-leader-based schemes that were discussed in Section 4.

Figures 5 (a), (b) and (c), compare the performance of the existing schemes used in MVAPICH2 with the multi-leader-based point-to-point hierarchical schemes. We can see that having multiple leaders within a node alleviates the bottleneck condition and it leads to better performance. On our hardware, since we have four sockets per node, it is optimal to assign four leader processes per node. We notice significant performance gains by using the 4-leader scheme in comparison with the single-leader-based scheme for small and medium sized messages. As explained in Section 4.1, we can see that the 8-leader scheme performs slightly worse than the 4-leader scheme. In the trivial case, the scheme with 16-leaders per node showed poor results.

In Figures 6 (a) and (b), we present the performance of the multi-leader-based shared memory approach. In both the graphs, we can clearly discern the advantage of using multiple leaders for the shared memory approach. As with the point-to-point-based scheme, we can observe the performance degradation by using 8-leaders per node.

In Section 4.2, we explained the significance of leader selection and the role it plays in the performance of our designs. To delineate this behavior, we consider two extreme cases. In one scenario, we will have all the leaders residing on one socket. In the other case, we allocate one leader per socket. Figure 7 compares the performance of these two CPU mapping schemes against the existing scheme. We can see that we begin to lose the advantages offered by the multi-leader-based designs if we choose the wrong CPU mapping scheme. This graph brings out the important issue of cache sharing that was addressed in [**?**].

5.4. Overall Comparison

In Figures 8 (a) and (b), we make a high level comparison by consolidating all the schemes that we have presented so far. For the multi-leader-based scheme, we only consider the scheme with 4-leaders as we know that the 4-leader scheme does better.

From Figure 8 (a), we can conclude that there is a 58% performance improvement with the 4-leader shared memory approach in comparison with the existing schemes in MVAPICH2 for small messages. In Figure 8 (b), we can see that this trend continues as the multi-leader-based shared memory scheme does about 70% better than the existing scheme used in MVAPICH2 at 4K data size. For larger messages, we noticed that the proposed multi-leader-based scheme does marginally worse(about 5%) than the existing schemes. We expect this to be the case as the hierarchical scheme involves making additional copies of the messages and this adds an undesirable overhead as the message size increases which counter-balances the benefits of the using multiple leaders. In Table 1, we summarize the different schemes that our integrated design uses. For small and medium sized messages, we make use of the proposed multi-leader-based hierarchical approaches. For large messages, we fall back to the *conventional* ring algorithm.

6. Related Work

Previously, several hierarchical approaches have been proposed for MPI_Allgather. In [**?**], [**?**], a single-leader-

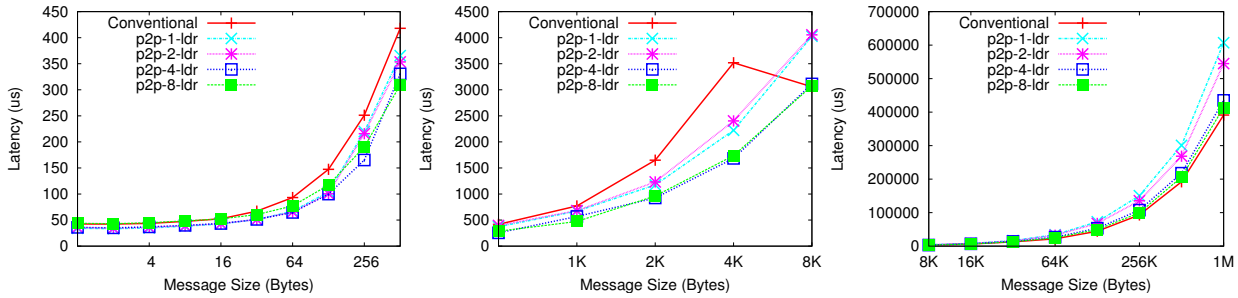

Figure 5. Performance of Multi-Leader-Based Point-to-Point Hierarchical Model for: (a) Small messages, (b) Medium messages and, (c) Large messages

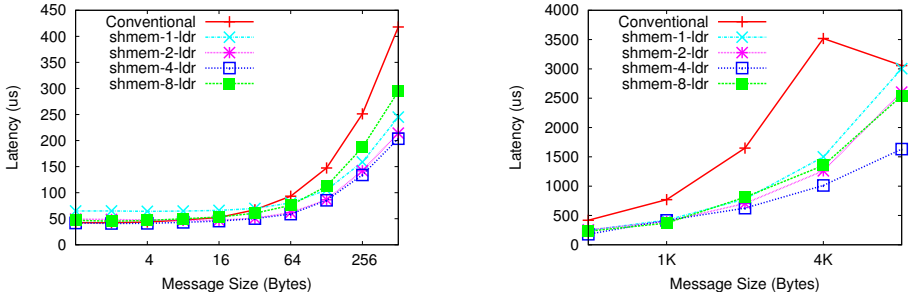

Figure 6. Performance of Multi-Leader-Based Shared Memory Hierarchical Model for: (a) Small messages and, (b) Medium messages

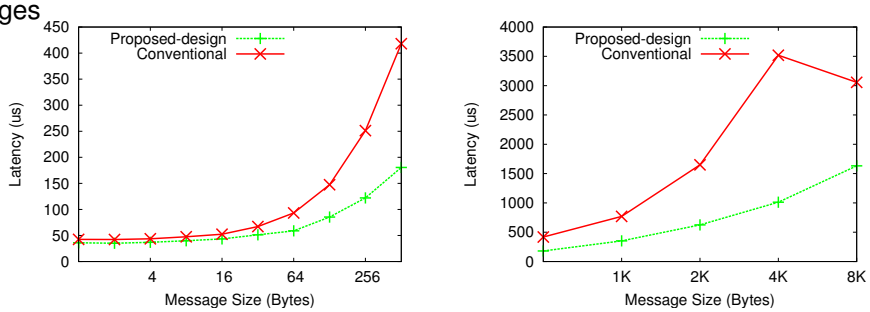

Figure 8. Overall Performance Comparison for: (a) Small messages and, (b) Medium messages

Table 1. Design Choice for Inter-Leader Exchange for Varying Message Sizes

Message Size	Intra-node exchange Mechanism	Inter-leader exchange algorithm	Design
Small	Point-to-Point	Recursive Doubling	Hierarchical
Medium	Shared Memory	Recursive Doubling	Hierarchical
Large	Point-to-Point	Ring	Conventional

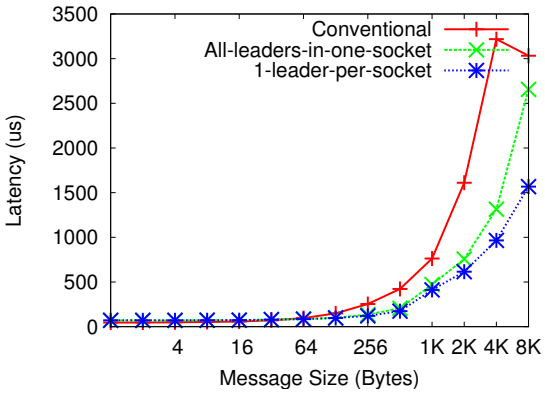

Figure 7. Impact on Performance with Various Leader Mapping Schemes

based hierarchical scheme has been proposed for SMP machines and the potential advantages of using an explicit shared memory buffer for intra-node exchanges have been described. In [**?**], the authors describe the role played by various parameters like affinity and cache behavior, in the context of multi-core architectures. They also present a clear picture regarding the manner in which the shared memory buffer needs to be used to achieve optimal performance for a few collectives on multi-cores. In [**?**], hierarchical approaches that utilize shared memory for intra-node exchanges have been proposed to improve the performance of collective operations. The authors also describe the impact of the block and cyclic distribution scheme of mapping

processes to cores on the performance of MPI_Allgather. This idea might appear to be trivial, but it has a strong impact on the behavior of the ring algorithm. Schemes have also been proposed to optimize the inter-node communication for Allgather, on InfiniBand clusters by using the RDMA techniques, in [**?**]. In our work, we highlight the limitations of these schemes and propose a novel multi-leader-based design to further improve the performance of the MPI_Allgather collective on multi-core systems.

7. Conclusion and Future Work

In this paper, we have addressed the need to have multi-core aware designs for the the MPI_Allgather routine to deliver high performance to applications that run on modern multi-core systems. We highlighted the specific factors that were leading to poor performance with the conventional schemes. We reviewed the existing hierarchical designs and pointed out the possible bottlenecks in those schemes. We proposed a novel multi-leader-based design to address these issues. With our new multi-leader-based schemes, we observe a performance improvement of about 60% for small messages and 70% for medium sized messages. As part of future work we are coming up with a framework for collective operations for choosing the optimal number of leaders for increasing core counts. We are also exploring alternatives to eliminate the multiple copies of messages that our current multi-leader-based hierarchical scheme uses.