Designing Multi-Leader-based AllGather Algorithms for Multicore Clusters *

K. Kandalla, H. Subramoni, G. Santhanaraman, M. Koop, D.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 computation cycles is being met by the use of multi-core processors. Having large number of cores per node necessitates the need for multi-core aware designs to extract the best performance. The Message Passing Interface (MPI) is the dominant parallel programming model on these clusters. MPI collective operations take a large 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 number of core per node increases. In this work,we propose a novel and more scalable multi-leader based Hierarchical AllGather design. This design allows better sharing for Non-Uniform Memory Access (NUMA) machines and makes better use of the network speed available with interconnects such as InfiniBand.The new multi-leader scheme achieves a performance improvement of up to 250% 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. Much of the additional computing power for these machines has been acheived through the use of multi-core processors. These additional cores per compute node add additional design choices into software design for high performance computing. At the same time, many clusters are designed using commodity high-performance interconnects, such as InfiniBand.

The Message Passing Interface (MPI) [6] is the dominant parallel programming model on these clusters. Given the role of the MPI library as the communication substrate for application communication, the library must be designed for multi-core clusters to achieve the highest performance. MPI defines two main types of communication: point-to-point and collective. The collective operations involve multiple tasks 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, overall performance of an application can improve.

Currently, optimized collectives for multi-core machines generally use the shared memory on the node to aggregate messages and have a single leader per node do 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* method for collectives. Instead of a single leader per node, we use a leader per socket on the compute node. This allows better sharing for Non-Uniform Memory Access (NUMA) machines and makes better use of the network speed available

*This research is supported in part by DOE grants #DE-FC02-06ER25749 and #DE-FC02-06ER25755; NSF grants #CNS-0403342 and #CCF-0702675.

with interconnects such as InfiniBand. We take one collective, `MPI_Allgather` and show how it can be transformed into a multi-leader approach. `MPI_Allgather` is an important collective and is used in applications like video compression [10] and matrix kernels.

We implement our design into MVAPICH2, a popular MPI library over InfiniBand. We have designed the hierarchical model of communication by considering both point-to-point schemes and a shared memory scheme. Using our design, we show a latency improvement of almost 200% for many message sizes including 16KB.

2 Background

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

Multi-core Architecture: In this section we give a brief background of common multi-core architectures. In this work we use the AMD multi-core platform for evaluating our proposed designs. Hence we give a brief description of this architecture. AMD multicore Opterons [3] are based on NUMA architecture with each of the sockets sharing independent memories. 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. Additionally, although current Intel-based systems are not NUMA, future designs will be to allow for increased memory bandwidth. The interconnect between sockets will be the Intel Quick Path Interconnect (QPI).

MPI_AllGather Collective: MPI_AllGather can be thought of as MPI_Gather but where all processes receive the result, instead of just the root.Thus in an AllGather collective operation each process supplies a vector of data which is broadcasted to everyone else. 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 800 organizations worldwide including several several high end computing platforms including the TACC Ranger system [11] which is currently ranked the sixth fastest supercomputer.

3 Existing Approaches for AllGather

In this section, we describe the various existing algorithms and their implementation in the MPI library.

AllGather algorithms: The most popular algorithms for MPI_Allgather are recursive doubling algorithms for short messages and ring algorithm for large messages[9]. All of these algorithms are currently employed in MPICH [7]. The recursive doubling algorithm consists of $log(n)$ steps where the data for each step doubles.

Point-to-Point based AllGather implementation: 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 MPICH implementation. 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. They also do not take advantage of shared memory for communicating data across cores residing on the same node.

The conventional algorithms described in the previous section do not take into account the hierarachy created due to multi-core architectures. These algorithms are ideally suited for single

core / single processor nodes. These approaches have severe limitations. In a worst case scenario, where a block distribution [1] 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 cyclic distribution of processes [1]

Hierarchical algorithms: Hierarchical approaches have been proposed for multicore clusters [1] to address this issue. In this approach one process is assigned as a 'leader' on each node and the other processes on that node are referred to as 'children'. Several approaches can be used to choose the leader. One simple approach would be to choose the lowest ranking process on each node as the leader. The entire algorithm could be broken up into three phases (i) Aggregation of messages at the leader processes (ii) Data exchange between leaders and (iii) Distribution of data from the leaders to chidren.

The hierarchical approaches can make use of the underlying point to point calls for transferring data within the node. For inter-node exchanges (phase-2), it has been shown [8] that recursive doubling scheme does better for smaller messages and the ring scheme is mode optimal for larger messages. For Intra-node exchanges there are potentially two approaches. (i) point-to-point and (ii)shared memory based.

In point-to-point approaches, the data transfers in phase-1 and phase-3 are implemented on top of point-to-point MPI_Send and MPI_Recv operations. Even if this phase is implemented with non-blocking calls, whether we have ring or the recursive doubling scheme for exchange within the node, the entire operation is still serialised. In the shared memory based approach, the leader sets up a shared memory buffer and all the other processes read/write their contibution into this shared memory space concurrently.

Even though the shared memory approach reduces serialization, the single leader scheme still has limitations that with increasing cores, more processes reading and writing from the same buffer would lead to memory contention. This would result in lower scalability for the single leader approach.

4 Design of Multi-Leader based AllGather Algorithms

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

Earlier work showed that a leader-based scheme could increase performance. From our analysis, however, we found that a single leader can have very high contention. As noted in Section 3, these can be implemented using point to point or shared memory. In the point to point design we suspect that the leader became a bottleneck in processing messages. In the shared memory design, we suspect that the memory contention between processes became significant.

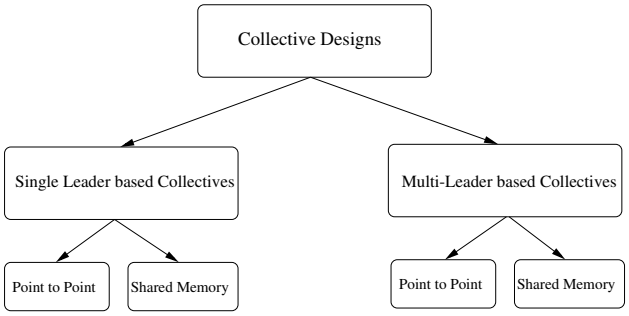


Figure 1: Collective design alternatives

As a result, we propose a *multi-leader* design to reduce the contention on memory and the leader process. Figure 1 shows the overall framework that is being designed. The left part of

the tree is the existing single-leader design. The right half of the tree is the design that we are proposing. There are a number of issues that arise when moving to a multi-leader design that we will cover in the following subsections.

4.1 Number of Leaders

The first major choice when designing a multi-leader collective is the number of leaders that each node should have. Existing designs have used only a single leader. In this case, when using a shared memory design all processes will attempt to read and write from a single buffer during the distribution and aggregation phases of the algorithm. When using a single leader with point-to-point aggregation and distribution, the single leader can become a bottleneck as well by having to process many messages. Increasing the number of leaders is a way to avoid such a problem. In particular, as more architectures become NUMA, sharing buffers across sockets in the system can lead to lower than expected performance. We propose using a single leader per socket to alleviate these problems.

4.2 Leader Selection

Once the number of leaders has been decided, the next design choice is to select leaders. Given our basis for desiring a multi-leader approach, memory contention is a key issue of concern. We opt to place leaders such that there is atmost 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.

4.3 Data Transfer

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 four schemes to aggregate messages at the leaders.

- Point to Point:
  - *ring exchange* between all the child processes that share the same leader.
  - *gather design* in which the leader process invokes a call to `MPI_Gather` with the communicator subgroup that comprises of all its children.
  - *explicit receive* design, where the leader executes calls to `MPI_Irecv` as each of its children send their contribution to the leader.
- 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 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 Leader Data Transfer

Each of the leaders must exchange data with all of the other leaders in the operation. In this phase 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 the tasks in the subgroup that are associated with it. If we have used a ring approach in the aggregation phase, all the children participate and at the end of the step, they 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 either the gather design or the explicit receive 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 design each child can simply read from the shared buffer.

5 Experimental Results

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

5.1 Experimental Testbed

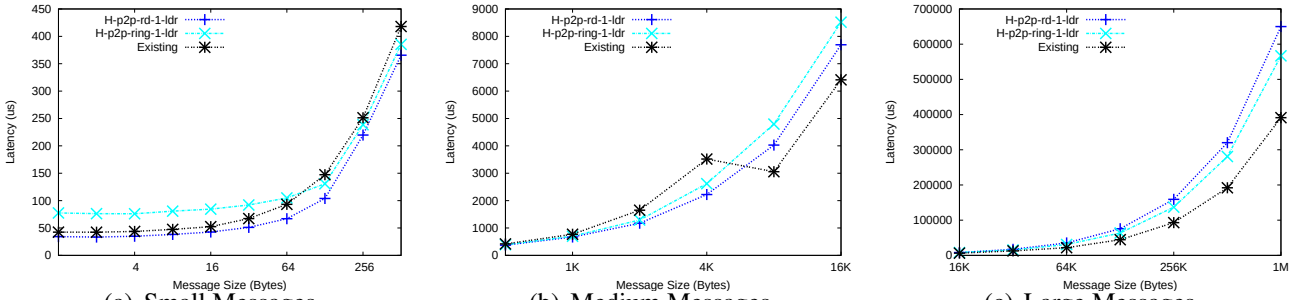
Each node of our testbed has 16 AMD Opteron 1.95 Ghz processors with 512 KB L2 cache. We ran our experiments on 4 such nodes with 64 processes. Each node also has 16 Gigabyte memory and PCI-Express bus. They are equipped with MT25418 HCAs with PCI-Ex interfaces. A 24-port Mellanox switch is used to connect all the nodes. The operating system used was RedHat Enterprise Linux Server 5.

Single Leader Scheme Evaluation: We first compare the base version of MVAPICH2 with the different single leader point-to-point Hierarchical model that we described in section 3. We will henceforth refer to this base version as '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 very little variation 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 Figure2.a we compare the performance of the single leader point-to-point approach with the conventional scheme for small messages. We can infer that there are some advantages of using a Hierarchical mode. As expected, we can observe that the scheme that uses recursive doubling for inter-node exchanges performs better for smaller messages. From Figure2.b and Figure2.c, we can observe that for medium sized messages,the schemes that use the ring algorithm for inter-leader exchange performs better. However, beyond a point, the conventional ring scheme starts out-performing the single leader point-to-point Hierarchical model. This could be because of the bottle-neck at the leader processes at each node.

In Figures 3.a and 3.b, we show the performance of the single-leader shared memory Hierarchical model. We can see that there are advantages of using the shared memory model for medium sized messages. However, for smaller messages the existing recursive doubling scheme performs better. As explained earlier, this could be attributed to the memory contention within each node. As expected, we could see some benefits of using the single leader Hierarchical approach. But, we could also identify the issues that were causing this approach to perform poorly at certain message sizes.

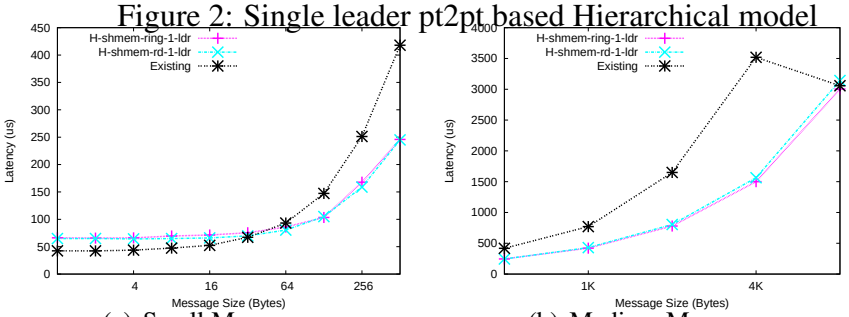
Multi-Leader Scheme Evaluation: In the following set of plots, we present the performance of the multi-leader schemes that were discussed in section 4.

Figures 4.a, 4.b and 4.c compare the performance of the existing schemes used in MVAPICH2 with the point-to-point based Multi-leader Hierarchical schemes. On observing the three figures,



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

Figure 2: Single leader pt2pt based Hierarchical model

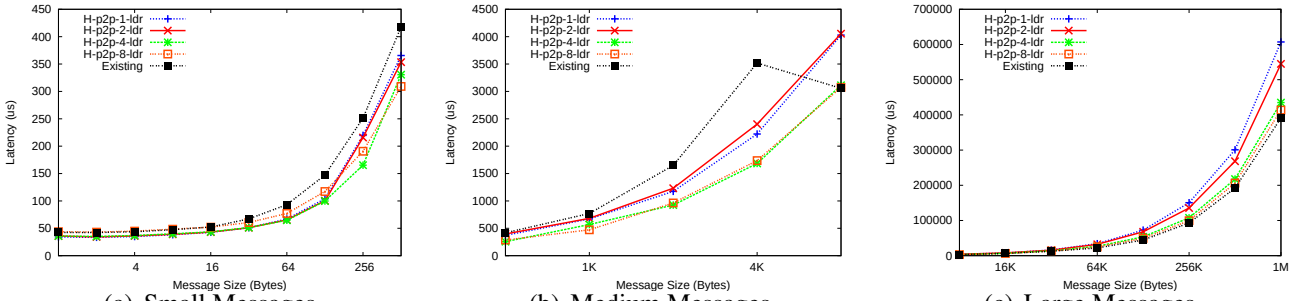


(a) Small Messages (b) Medium Messages

Figure 3: Single leader shared memory based Hierarchical model

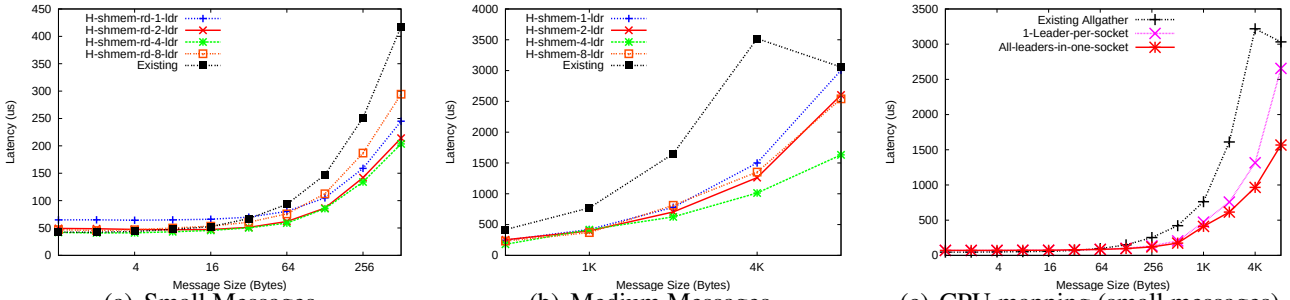
we can infer that the model with 2-leaders does better than a single leader model for large messages and the 4-leader model does better than the 2-leader model for most data sizes. However, we can see that the performance of the 8-leader model is slightly worse that the 4-leader model for smaller messages and we expect the 16-leader model to do much worse.

In Figures 5.a and 5.b, we present the performance of the shared memory based multi-leader approach. In both the plots, we can clearly discern the advantage of using more than 1-leader for the shared memory approach. However, we can also observe the performance degradation with the 8-leader scheme. 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 behaviour, 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 5.c compares the performance of these two CPU mapping schemes against the base allgather scheme. We can see that we begin to lose the advantages offered by the hierarchical shared memory designs if we choose the wrong CPU mapping scheme.



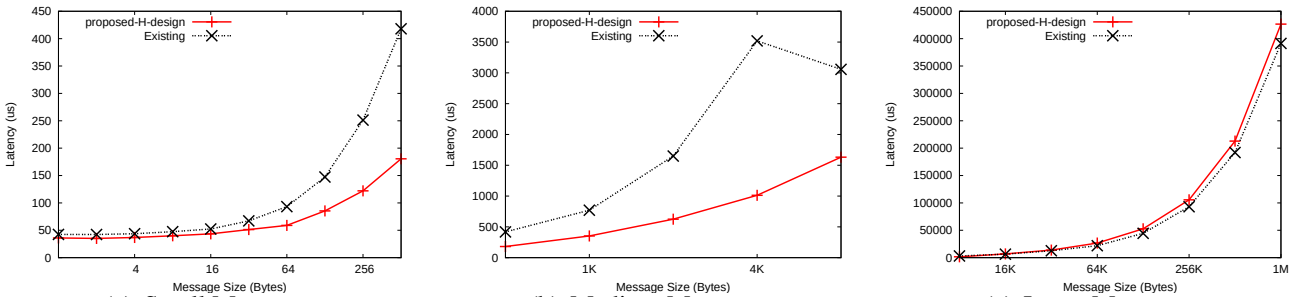
(a) Small Messages (b) Medium Messages (c) Large Messages

Figure 4: Multi leader pt2pt based Hierarchical model



(a) Small Messages (b) Medium Messages (c) CPU mapping (small messages)

Figure 5: (a) Multi leader shared memory based Hierarchical model and (b) CPU Mapping schemes



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

Figure 6: Overall comparison

Overall Comparison: In Figures 6.a, 6.b and 6.c, we make a high level comparison by consolidating all the schemes that we have presented so far. For the multi-leader scheme, we only consider the scheme with 4-leaders as we know that the 4-leader scheme does better. Our integrated design utilizes point-to-point based Multi-leader Hierarchical scheme with recursive doubling for inter-leader exchange for very small messages, shared memory based Multi-leader Hierarchical approach for medium messages. We again switch back to point-to-point based Multi-leader Hierarchical approach with the ring scheme for inter-leader exchanges for large messages. From Figure 6.a, we can conclude that there is a 200% performance improvement with the 4-leader shared memory approach in comparison with the existing schemes in MVAPICH2 for small messages. In Figure 6.b, we can see that this trend continues as the multi-leader shared memory scheme does about 250% better than the existing scheme used in MVAPICH2 at 4K data size. In Figure 6.c, we present the comparison of the 4-leader point-to-point scheme with the existing scheme. We can see that the new scheme does only marginally worse(about 5%) than the existing schemes. We can consider this mild variation to be within experimental error margin.

6 Related Work

Previously, single leader hierarchical approaches have been proposed for SMP systems [5]. This work discusses the potential advantages of using the shared memory scheme for small and medium sized messages instead of the point-to-point approach. In [4],the authors propose multi-core aware methods for some of the collective operations.In [1], the shared memory concept in the multi-core systems have been exploited to optimize the intra-node phase of the Allgather communication. The paper also describes how the inter-node communication can be optimized by using the RDMA schemes that are offered by the modern infiniband interconnects. In [2] the authors describe the impact of the block and cyclic distribution scheme of mapping processes to cores on the perfor-

mance of collectives. Our approach further extends some of these shared memory designs as well as propose a novel multi-leader design to alleviate the limitations of these approaches.

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 design to address these issues. With our new multi-leader schemes, we observe a performance improvement of almost 250% for medium sized messages. For future work we are planning to come up with a framework for collective operations for choosing the optimal number of leaders for increasing core counts.

References

[1] D. D. A. Mamidala, R. Kumar and D. K. Panda. Mpi collectives on modern multicore clusters: Performance optimizations and communication characteristics. In *Int'l Symposium on Cluster Computing and the Grid (CCGrid), Lyon, France, May 2008*.

[2] A. R. Mamidala, A. Vishnu, D. K. Panda. Efficient Shared Memory and RDMA based design for MPI_Allgather over InfiniBand .

[3] AMD. http://www.AMD.com/opteron.

[4] R. L. Graham and G. M. Shipman. Mpi support for multi-core architectures: Optimized shared memory collectives. In *EuroPVM/MPI 2008*.

[5] Jesper Larsson Trff. Efficient Allgather for Regular SMP-Clusters .

[6] MPI Forum. MPI: A Message Passing Interface. In *Proceedings of Supercomputing*, 1993.

[7] MPICH2: High Performance portable MPI implementation. http://www.mcs.anl.gov/research/projects/mpich2.

[8] Rajeev Thakur, Rolf Rabenseifner, and William Gropp. Optimization of Collective Communication Operations in MPICH.

[9] Rajeev Thakur William Gropp. Improving the Performance of Collective Operations in MPICH.

[10] A. Rodriguez, A. Gonzalez, and M. Malumbres. Hierarchical Parallelization of an H.264/AVC Video Encoder. *Parallel Computing in Electrical Engineering, 2006. PAR ELEC 2006. International Symposium on*, pages 363–368, 2006.

[11] TACC Stampede Cluster. http://www.tacc.utexas.edu/resources/hpcsystems/.