Performance Modeling for Self Adapting Collective Communications for MPI*

Sathish S. Vadhiyar, Graham E. Fagg, and Jack J. Dongarra

Computer Science Department
University of Tennessee, Knoxville
{vss, fagg, dongarra}@cs.utk.edu

Abstract. The performance of the MPI's collective communications is critical in most MPI-based applications. A general algorithm for a given collective communication operation may not give good performance on all systems due to the differences in architectures, network parameters and the storage capacity of the underlying MPI implementation. Hence, collective communications have to be tuned for the system on which they will be executed. In order to determine the optimum parameters of collective communications on a given system in a time-efficient manner, the collective communications need to be modeled efficiently. In this paper, we discuss various techniques for modeling collective communications. We also discuss a dynamic topology method that uses the tuned static topology shape, but re-orders the logical addresses to compensate for changing run time variations.

1 Introduction

There have been a number of attempts in the past to improve the performance of the MPI collective communications for a given system. They either dealt with the collective communications for a specific system or tried to tune the collective communications for a given system based on mathematical models or both. Lars Paul Huse's paper on collective communications [2] studied and compared the performance of different collective algorithms on SCI based clusters. MAGPIE by Thilo Kielman et. al. [1] optimizes collective communications for clustered wide area systems. Though MAGPIE tries to find the optimum buffer size and optimum tree shape for a given collective communication on a given system, these optimum parameters are determined using a performance model called the parameterized LogP model. The MAGPIE model considered only a few network parameters for modeling collective communications. For example, it did not take into account the number of previously posted non-blocking sends, Isends, in determining the network parameters for a given message size.

In our previous work [13], [14], we built efficient algorithms for different collective communications and selected the best collective algorithm and segment size for a given {collective communication, number of processors, message

* This work was supported by the US Department of Energy through contract numberDE-FG02-99ER25378.

size} tuple by experimenting with all the algorithms and all possible values for message sizes. The tuned collective communication operations were compared with various native vendor MPI implementations. The use of the tuned collective communications resulted in about 30%-650% improvement in performance over the native MPI implementations. The tuning system uses the native MPI point to point sends and receives and does not take advantage of any lower-level communications like hardware-level broadcast etc.

Although efficient, conducting the actual set of experiments to determine the optimum parameters of collective communications for a given system, was found to be time-consuming. As a first step, the best buffer size for a given algorithm for a given number of processors was determined by evaluating the performance of the algorithm for different buffer sizes. In the second phase, the best algorithm for a given message size was chosen by repeating the first phase with a known set of algorithms and choosing the algorithm that gave the best result. In the third phase, the first and second phase were repeated for different number of processors. The large number of buffer sizes and the large number of processors significantly increased the time for conducting the above experiments.

In order to reduce the time for running the actual set of experiments, the collective communications have to be modeled effectively. In this paper, we discuss the various techniques for modeling the collective communications. The reduction of time for conducting actual experiments is achieved at 3 levels. In the first level, limited number of {collective communications, number of processors, message size} tuple combinations is explored. In the second level, the number of {algorithm, segment size} combinations for a given {collective communication, number of processors, message size} tuple is reduced. In the third level, the time needed for running an experiment for a single {collective communications, number of processors, message size, algorithm, segment size} tuple is reduced by modeling the actual experiment.

In Section 2, we give a brief overview of our previous work regarding the automatic tuning of the collective communications. We illustrate the automatic tuning with the broadcast communication. The results in Section 2 reiterate the usefulness of the automatic tuning approach. These results were obtained by conducting the actual experiments with all possible input parameters. In Section 3, we describe three techniques needed for reducing the large number of actual experiments. In Section 4, we discuss the dynamic topology method that reorders the processes within a given topology for communication. In Section 5, we present some conclusions. Finally in Section 6, we outline the future direction of our research.

2 Automatically Tuned Collective Communications

A crucial step in our effort was to develop a set of competent algorithms. Table. 1 lists the various algorithms used for different collective communications.

While there are other more competent algorithms for collective communications, the algorithms shown in Table. 1 are some of the most commonly used

Table 1. Collective communication algorithms

Collective Communications	*Algorithms*
Broadcast	Sequential, Chain, Binary and Binomial
Scatter	Sequential, Chain and Binary
Gather	Sequential, Chain and Binary
Reduce	Gather followed by operation, Chain, Binary, Binomial and Rabenseifner
Allreduce	Reduce followed by broadcast, Allgather followed by operation, Chain, Binary, Binomial and Rabenseifner
Allgather	Gather followed by broadcast
Allgather	Circular
Barrier	Extended ring, Distributed binomial and tournament

algorithms. For algorithms that involve more than one collective communication (e.g., reduce followed by broadcast in allreduce), the optimized versions of the collective communications were used. The segmentation of messages was implemented for sequential, chain, binary and binomial algorithms for all the collective communication operations.

2.1 Results For Broadcast

The experiments consist of many phases.

Phase 1: Determining the best segment size for a given {collective operation, number of processors, message size, algorithm} tuple. The segment sizes are powers of 2, multiples of the basic data type and less than the message size.

Phase 2: Determining the best algorithm for a given {collective operation, number of processors} tuple for each message size. Message sizes from the size of the basic data type to 1MB were evaluated.

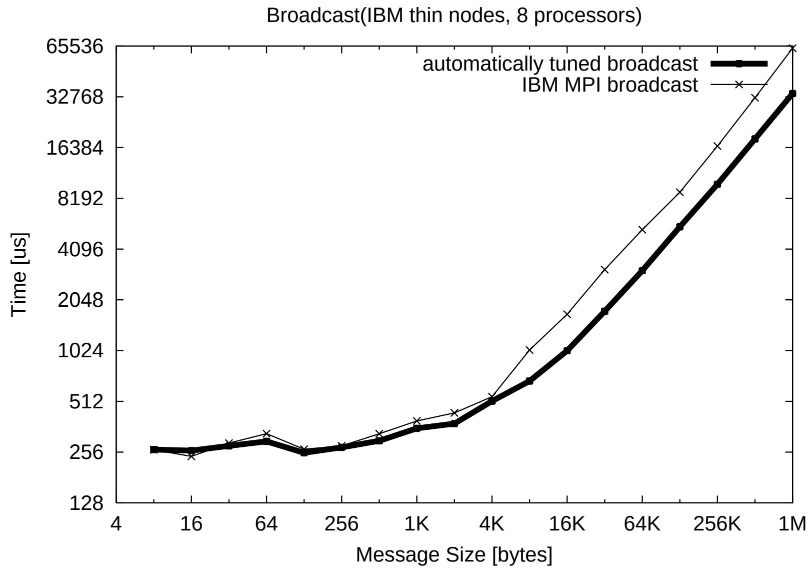
Phase 3: Repeating phase 1 and phase 2 for different {number of processors, collective operation} combinations. The number of processors will be power of 2 and less than the available number of processors.

Our current effort is in reducing the search space involved in each of the above phases and still be able to get valid conclusions.

The experiments were conducted on four different classes of systems, including Sparc clusters and Pentium workstations and two different types of PowerPC based IBM SP2 nodes.

Fig. 1 shows the results for a tuned MPI broadcast on an IBM SP2 using "thin" nodes that are interconnected by a high performance switch with a peak bandwidth of 150 MB/s verses the IBM optimized vendor MPI implementation. Similar encouraging results were obtained for other systems as detailed in [12] & [13].

Fig. 1. Broadcast Results (IBM thin nodes)



3 Reducing the Number of Experiments

In the experimental method mentioned in the previous section, about 13000 individual experiments have to be conducted. Even though this only needs to occur once, the time taken for all these experiments was considerable and was approximately equal to 50 hours.

The experiments conducted consist of two stages, the primary set of steps is dependent on message size, number of processors and MPI collective operation, i.e. the tuple {message size, processors, operation}. For example 64KBytes of data, 8 process broadcast. The secondary set of tests is an optimization at these parameters for the correct method (topology-algorithm pair) and segmentation size, i.e. the tuple {method, segment size}.

Reducing the time needed for running the actual experiments can be achieved at three different levels:

1. reducing the primary tests
2. reducing the secondary tests and
3. reducing the time for a single experiment, i.e. for a single {message size, processors, operation, method, segment size} instance.

3.1 Reducing the Primary Tests

Currently the primary tests are conducted on a fixed set of parameters, in effect making a discrete 3D grid of points. For example, varying the message size in powers of two from 8 bytes to 1 MByte, processors from 2 to 32 and the MPI operations from Broadcast to All2All etc.

This produces an extensive set of results from which accurate decisions will be made at run-time. This however makes the initial experiments time consuming and also leads to large lookup tables that have to be referenced at run time, although simple caching techniques can alleviate this particular problem.

Currently we are examining three techniques to reduce this primary set of experimental points.

1. Reduced number of grid points with interpolation. For example reducing the message size tests from {8, 16, 32, 64.. 1MB} to {8, 1024, 8192.. 1MB}.
2. Using instrumented application runs to build a table of only those collective operations that are required, i.e. not tuning operations that will never be called, or are called infrequently.
3. Using combinatorial optimizers with a reduced set of experiments, so that complex non-linear relationships between points can be correctly predicted.

3.2 Reducing the Secondary Tests

The secondary set of tests for each {message size, processors, operation} tuple are where we have to optimize the time taken, by changing the method used (algorithm/topology) and the segmentation size (used to increase the bi-sectional bandwidth utilization of links), i.e. {method, segment size}. Fig. 2 shows the performance of four different methods for solving an 8 processor MPI Scatter of 128KBytes of data on a Sparc cluster. Several important points can be observed. Firstly, all the methods have the same basic shape that follows the form of an exponential slope followed by a plateau. Secondly, the results have multiple local optima, and that the final result (segment size equal to message size) is not usually the optimal but is close in magnitude to the optimal.

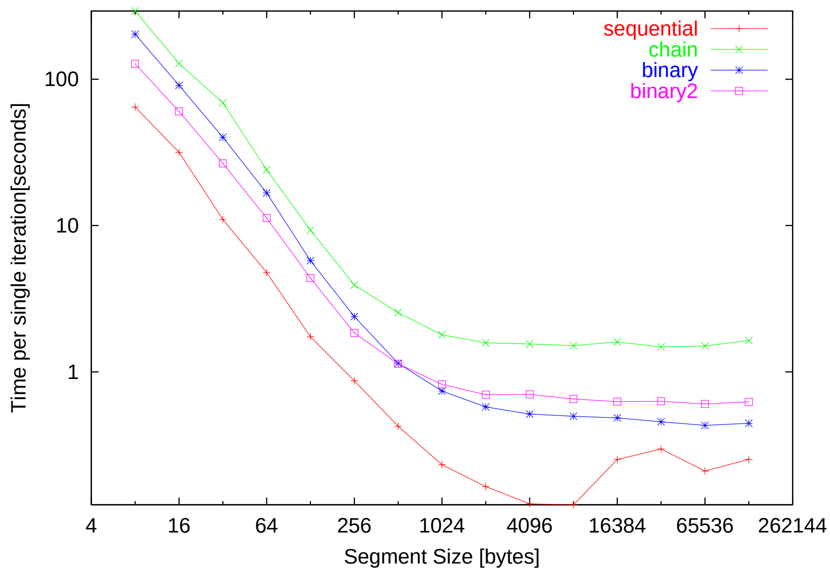
The time taken per iteration for each method is not constant, thus many of the commonly used optimization techniques cannot be used without modification. For example in Fig. 2, a test near the largest segment size is in the order of hundreds of microseconds whereas a single test near the smallest segment size can be in the order of a 100 seconds, or two to three orders of magnitude larger.

For this reason we have developed two methods that reduce the search space to tests close to the optimal values, and a third that runs a full set of segment-size tests on only a partial set of nodes.

The first two methods use a number of different hill descent algorithms known as the Modified Gradient Descent MGD and the Scanning Modified Gradient Descent (SMGD) that are explained in [13]. They primarily reduce the search times by searching the least expensive (in time) search spaces first while performing various look ahead algorithms to avoid non optimal minima. Using these two methods the time to find the optimal segment size for the scatter shown in Fig. 2 is reduced from 12613 seconds to just 39 seconds or a speed up of 318.

The number of segment sizes to explore can also be reduced by considering certain characteristics of the architecture. For example, in some architectures,

Fig. 2. Segment size verse time for various communication methods



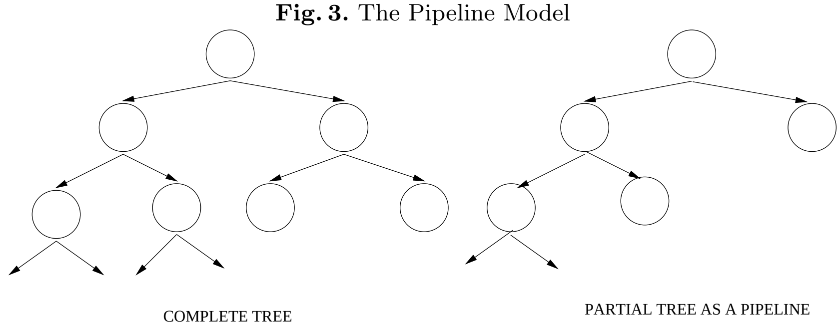
the size of the packets used in communications is known beforehand. Hence the optimum segment size in these architectures will be within a certain range near the packet size used for communications.

The third method used to reduce tests is based on the relationship between some performance metrics of a collective that utilizes a tree topology and those of a pipeline that is based only on the longest edge of the tree as shown in Fig. 3. In particular the authors found that the pipeline can be used to find the optimal segmentation size at greatly reduced time as only a few nodes need to be tested as opposed to the whole tree structure. For the 128 KB 8 process scatter discussed above, an optimal segment size was found in around 1.6 seconds per class of communication method (such as tree, sequential or ring). i.e. 6.4 seconds verses 39 seconds for the gradient descent methods on the complete topologies or 12613 seconds for the complete exhaustive search.

3.3 Reducing the single-experiment time

Running the actual experiments to determine the optimized parameters for collective communications is time-consuming due to the overheads associated with the startup of different processes, setting up of the actual data buffers, communication of messages between different processes etc.. We are building experimental models that simulate the collective algorithms but incur less time to execute than the actual experiments. Since the collective communication algorithms are based on the MPI point to point sends and receives and do not use any lower level communications, the models for collective communications do not take into account

Fig. 3. The Pipeline Model



the raw hardware characteristics like the link bandwidth, latency, topology etc. Instead they take into account times for MPI point to point communications like the send overhead, receive overhead etc. As part of this approach, we discuss the modeling experiments for broadcast in the following sub sections.

General Overview All the broadcast algorithms are based on a common methodology. The root in the broadcast tree continuously does non-blocking sends of MPI, MPI_Isends, to send individual message buffers to its children. The other nodes post all their non-blocking receives of MPI, MPI_Irecvs, initially. The nodes between the root node and the leaf nodes in the broadcast tree, send a segment to their children as soon as the segment is received.

After determining the times for individual Isends and the times for message receptions, a broadcast schedule as illustrated by Fig. 4 can be used to predict the total completion time for the broadcast.

A broadcast schedule such as the one shown in Fig. 4 can be used to accurately model the overlap in communications, a feature that was lacking in the parameterized LogP model [1].

Measurement of Point to Point Communications As observed in the previous section, accurate measurements of the time for Isends and the time for the reception of the messages are necessary for efficient modeling of broadcast operations. Previous communications models [3], [1], do not efficiently take into account the different types of Isends. Also, these models overlook the fact that the performance of an Isend can vary depending on the number of Isends posted previously. Thus the parameters, the send overhead, *os(m)*, the receive overhead, *or(m)*, the gap value, *g(m)*, for a given message size m, that were discussed in the parameterized LogP model can vary from a particular point in execution to another depending on the number of pending Isends and the type of the Isend.

MPI implementations employ different types of Isends depending on the size of the message transmitted. The popular modes of Isends are blocking, immediate and rendezvous and are illustrated by Fig. 5

Fig. 4. Illustration of Broadcast Schedule

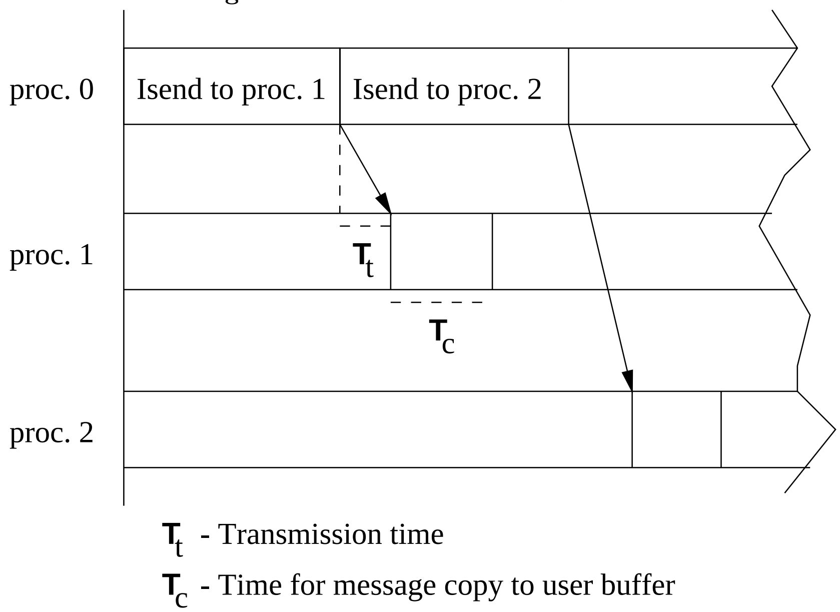
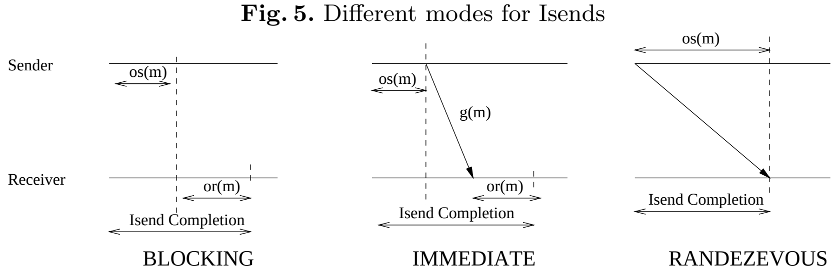


Fig. 5. Different modes for Isends



The parameters associated with the different modes of Isends can vary depending on the number of Isends posted earlier. Hence, for example, in the case of immediate mode, the Isends can lead to overflow of buffer space in the receive end, which will eventually result in larger g(m) and os(m).

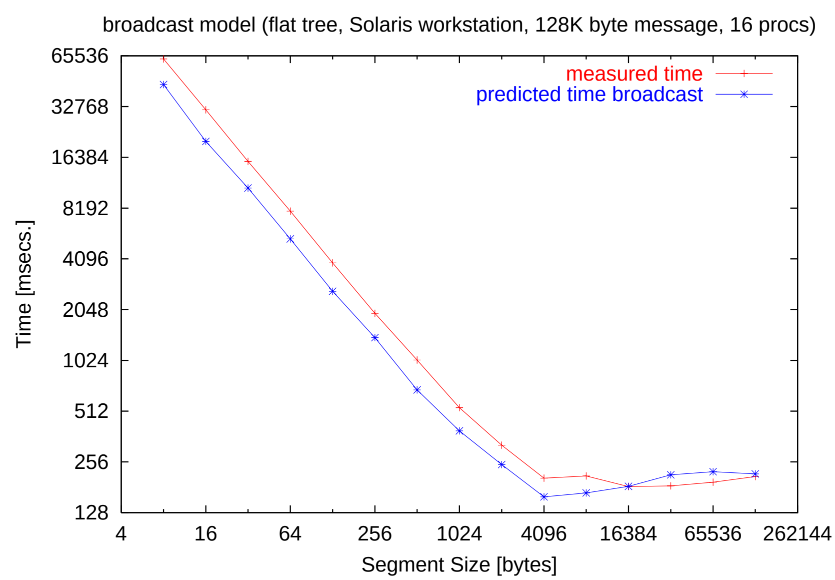
Model Based on Communication Schedules In this section, we describe a simple model that we have built to calculate the performance of collective communications. The model is based on point-point communication times and uses communication schedules for collective operations similar to the broadcast schedule shown in Fig. 4.

The model uses the data for sender overhead, *os(m)*, receiver overhead, *or(m)* and gap value, *g(m)* for the different types of Isends show in Fig. 5. The send overhead, *os(m)* is determined for different message sizes by observing the time taken for the corresponding Isends. The time for Isends, *os(m)*, increases as the message size is increased up to a certain message size beyond which, os(m), falls to a small value. At this message size, the Isend switches from the blocking to immediate mode. *or(m)* for blocking mode is determined by allowing the receiver to post a blocking receive after making sure the message has been transmitted over the network to the receiver end and determining the time taken for the blocking receive. In the immediate mode, the sender has to wait for *g(m)* before transmitting the next message. This time is determined by posting an Isend and determining the time taken for the subsequent Wait. In the immediate mode, *or(m)* is calculated by calculating *or(m)+g(m)*. *or(m)+g(m)* is calculated by determining the time for a ping-pong transmission between a sender and a receiver and subtracting 2**os(m)* from the ping-pong time. For each of the above experiments, 10 different runs were made and averages were calculated. The experiments were repeated at different points in time on shared machines and the standard deviation was found to be as low as 40.

The following results illustrate the prediction accuracy of the models. While the experiments were conducted only on a Sparc cluster, similar experiments will be conducted on other systems to validate the predication accuracy on other systems.

Fig. 6 compares the actual and predicted broadcast times for a flat tree broadcast sending a 128K byte message using 8 processors on a Sparc cluster.

Fig. 6. Flat Tree broadcast



We can observe that the predicted times are close to the actual broadcast times. According to the predicted results, the optimum segment size for the flat tree broadcast for 128KB message size is 4KB, whereas, according to the actual times, the optimum segment size is 16KB. But the ratio between the actual time at 4KB and the actual time at 16KB is found to be just 1.12.

Fig. 7 compares the actual and predicted broadcast times for a binary tree broadcast and Fig. 8 compares the actual and predicted broadcast times for a binomial tree broadcast. In these cases, the ratios between the actual times for predicted and actual optimum segment sizes were found to be 1.09 and 1.01 respectively.

Fig. 7. Binary Tree broadcast

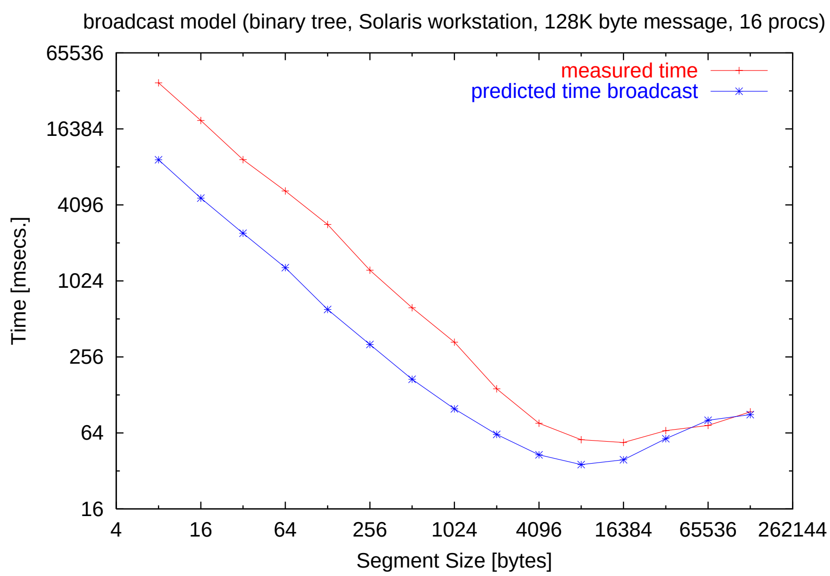


Fig. 9 shows the actual and predicted values of the various broadcast algorithms for 128K byte message size. Comparison of the relative performance of the broadcast algorithms using actual and predicted results leads to the same conclusions for broadcast, i.e., flat tree gives the worst performance and binary tree gives optimum performance. Thus we find that the model is able to predict both optimum segment sizes within a single algorithm and optimum algorithms when comparing different algorithms.

While models for other important collective communications like scatter and gather are not implemented, modeling the other collective communications is similar to modeling broadcast with few additional issues.

A scatter operation is similar to broadcast operation except that the sender has to make strides in the send buffer to send the next element to its next

Fig. 8. Binomial Tree broadcast

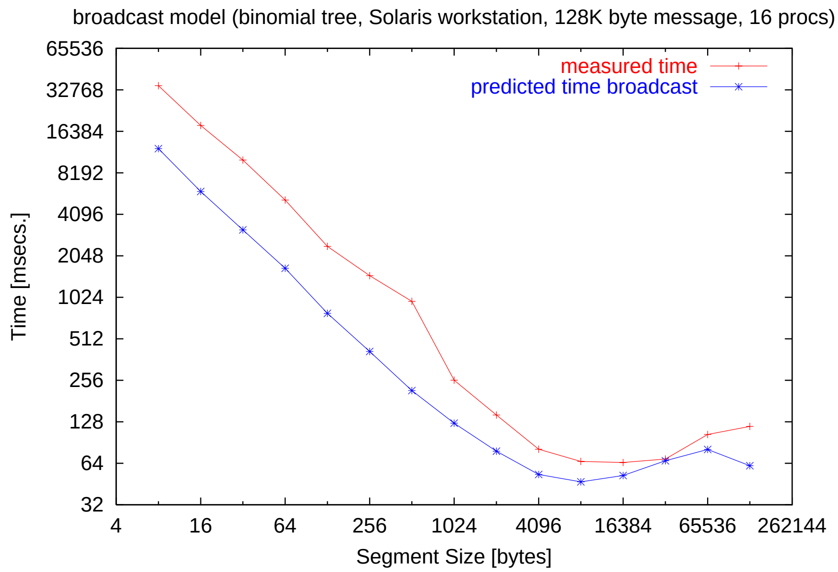
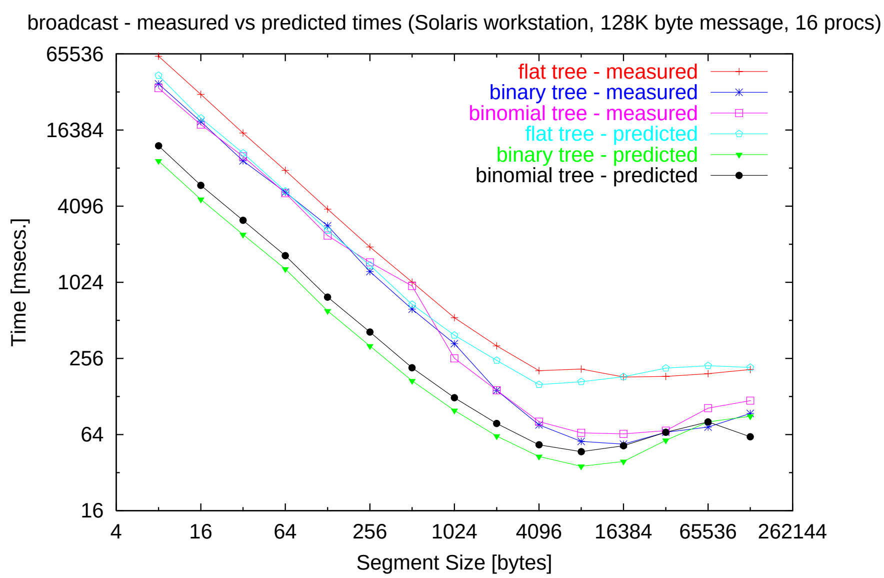


Fig. 9. Comparison of algorithms - Measured vs Predicted times



child. For small buffer sizes, the entire buffer is brought inside the cache and our broadcast model should be applicable to scatter as well. For large buffer sizes, additional complexity is introduced due to frequent cache misses. In that case our model needs to take into account the time needed for bringing data from memory to cache and compare this time with the gap time for the previous Isend.

Modeling gather is more challenging than modeling broadcast or scatter since three different scenarios have to be considered. For small buffer sizes, the time for receive of a segment by the root assuming the children have already posted their sends have to be modeled and techniques used in modeling broadcast and scatter can be used. For large buffer sizes, issues regarding movement of data from memory to cache also apply to gather and the corresponding techniques used for scatter can be used. For large number of segments, the children of the root will be posting large number of Isends to the same destination, i.e. the root. In this case, the storage of pending communications will get exhausted, and the performance of Isends will deteriorate. Some benchmark tests can be performed before hand to determine the point when the performance of Isends degrades and can be plugged into the model.

Models for other collective communications like allreduce, allgather etc. can be built based on the experience of modeling broadcast, scatter and gather.

Although the models are primarily used to reduce the single-experiment time, they can also be used to reduce the number of segment sizes to explore. For example, in architectures where fixed size packets are used for communications, the send and receive overheads for large message sizes will be approximately multiples of the overhead times associated with the message of size equal to the packet size. Hence simulation experiments can only be conducted for those segment sizes close to the packet size.

4 Dynamic Reordering of Topologies

Most systems rely on all processes in a communicator or process group entering the collective communication call synchronously for good performance, i.e. all processes can start the operation without forcing others later in the topology to be delayed. There are some obvious cases where this is not the case:

1. The application is executed upon heterogeneous computing platforms where the raw CPU power varies (or load balancing is not optimal).
2. The computational cycle time of the application can be non-deterministic as is the case in many of the newer iterative solvers that may converge at different rates continuously.

Even when the application executes in a regular pattern, the physical network characteristics can cause problems with the simple LogP model, such as when running between dispersed clusters. This problem becomes even more acute when the system latency is so low, that any buffering, while waiting for slower nodes, drastically changes performance characteristics as is the case with BIP-MPI [8].

4.1 Dynamic Methodology

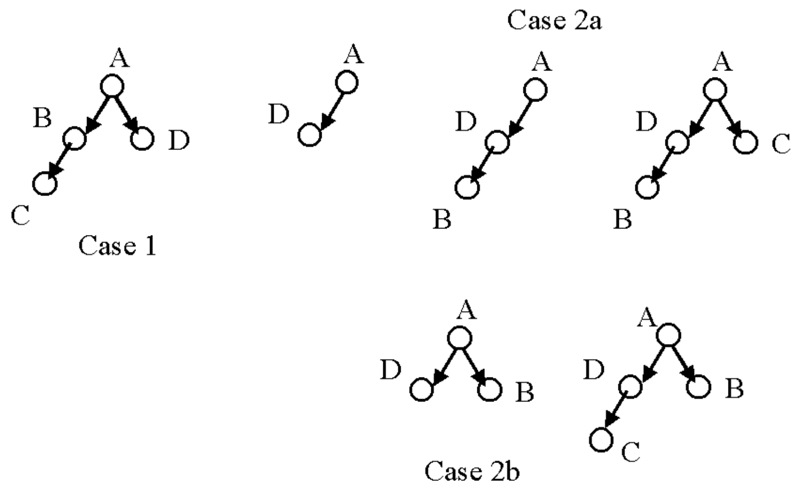
This method is a modification of the previous tuned method, where we use the tuned topology as a starting point, but the behavior of the method is varied between actual uses of the collective operations at run-time. The method forces all the non-root nodes to send a small start-acknowledge (SACK) message to the root node, which the root uses to build a mapping from communicator rank to logical address within the chosen topology dynamically. Each process, after having sent its SACK, then receives its own topology information via the root directly or by piggy backing the information on a user data message depending on the MPI operation being performed. This information can be split into multiple messages such as from whom do they receive from, and whom do they send to, as the information becomes available. i.e. a process might not be a leaf node in the tree topology but still receives all its data before knowing whom to send to.

Fig. 10 demonstrates this methodology. Case 1 is where all processes within the tree are ready to run immediately and thus performance is optimal. In Case 2, both processes B and C are delayed and initially the root A can only send to D. As B and C become available, they are added to the topology. At this point we have to choose whether to add the nodes depth first as in Case 2a or breadth first as in Case 2b. Currently depth first has given us the best results. Also note that in CASE 1, if process B is not ready to receive, it affects not only its own sub-tree, but depending on the message/segment size, it is possible that it would block any other messages that A might send, such as to D's sub-tree etc. Faster network protocols might not implement non-blocking sends in a manner that could overcome this limitation without effecting the synchronous static optimal case, and thus blocking sends are often used instead.

Currently we are testing the cost of overhead incurred in using this technique for different network infrastructures. We are also exploring the conditions needed for the automatic use of this technique during the course of the computation. Initial results have been promising, especially for large messages and network interfaces with very low latency, that rely on the receivers to have already posted receives to allow DMA message transfers. Worst case results have been equivalent to the overhead for n-1 small message send/receives. Best case has been within a few percent of optimal where no re-ordering on the same example has produced multiples of the optimal wall clock times, although this varies with the operation, number of processors, data size and level of initial synchronization.

The re-ordering of topologies was tested using an 8 processor 1 MByte broadcast where several of the processes were delayed on entering the collective operation by 200, 300 and 400 milliseconds. The time for a non-delayed broadcast was around 425 milliseconds. The uncorrected broadcast took 896 milliseconds. The corrected topologies took 702 milliseconds for breadth first and 673 milliseconds for the depth first, representing a 22% and 25% improvement over the uncorrected topology.

Fig. 10. Reordering a tree topology



5 Conclusion

Modeling the collective communications to determine the optimum parameters of the collective communications is a challenging task, involving complex scenarios. A single simplified model will not be able to take into account the complexities associated with the communications. A multi-dimensional approach towards modeling, where various tools for modeling are provided to the user to accurately model the collective communications on his system, is necessary. Our techniques regarding the reduction of number of experiments are steps towards constructing the tools for modeling. These techniques have given promising results and have helped identify the inherent complexities associated with the collective communications. We have also shown that during application execution, dynamically altering the mapping between rank and position within a topology can yield additional benefits in terms of performance.

6 Future Work

While our initial results are promising and provide us some valuable insights regarding collective communications, much work still has to be done to provide comprehensive set of techniques for modeling collective communications. Selecting the right set of techniques for modeling based on the system dynamics is an interesting task and will be explored further.

References

1. Thilo Kielmann, Henri E. Bal and Segei Gorlatch. Bandwidth-efficient Collective Communication for Clustered Wide Area Systems.*IPDPS 2000*, Cancun, Mexico, May 1-5, 2000.
2. Lars Paul Huse. Collective Communication on Dedicated Clusters of Workstations.*Proceedings of the 6th European PVM/MPI Users' Group Meeting*, Barcelona, Spain, LNCS 1697, pp. 469-476, Springer, September 1999.
3. David Culler, R. Karp, D. Patterson, A. Sahay, K.E. Schauser, E. Santos , R. Subramonian and T. von Eicken. LogP: Towards a Realistic Model of Parallel Computation. In *Proc. Symposium on Principles and Practice of Parallel Programming*, pages 1-12, San Diego, CA., May 1993.
4. R. Rabenseifner. A new optimized MPI reduce algorithm. http://www.hlrs.de/structure/support/parallel_computing/models/mpi/myreduce.html (1997).
5. Marc Snir, Steve Otto, Steven Huss-Lederman, David Walker and Jack Dongarra. MPI- The Complete Reference. *Volume 1, The MPI Core, second edition* (1998).
6. M. Frigo. FFTW: An Adaptive Software Architecture for the FFT. *Proceedings of the ICASSP Conference*, page 1381, Vol. 3. (1998).
7. R. Clint Whaley and Jack Dongarra. Automatically Tuned Linear Algebra Software. *SC98: High Performance Networking and Computing.* http://www.cs.utk.edu/ rwhaley/ATL/INDEX.HTM. (1998).
8. L. Prylli and B. Tourancheau. BIP: a new protocol designed for high performance networking on myrinet. In the *PC-NOW workshop, IPPS/SPDP 1998*, Orlando, USA, 1998.
9. Debra Hensgen, Raphael Finkel and Udi Manber. Two algorithms for Barrier Synchronization. *International Journal of Parallel Programming*, Vol. 17, No. 1, 1988.
10. M. Beck, J. Dongarra, G. Fagg, A. Geist, P. Gray, J.Kohl, M. Migliardi, K. Moore, T. Moore, P. Papadopoulous, S. Scott, V. Sunderam. HARNESS: a next generation distributed virtual machine, *Journal of Future Generation Computer Systems*, (15), Elsevier Science B.V., 1999.
11. Graham E. Fagg, Antonin Bukovsky, and Jack J. Dongarra, "HARNESS and fault tolerant MPI", *Journal of Parallel Computing*, 27(11), Elsevier Science B.V., 2001.
12. Graham E. Fagg and Jack J. Dongarra. "FT-MPI: Fault Tolerant MPI, Supporting Dynamic Applications in a Dynamic World", *Proc. of EuroPVM-MPI 2000, Lecture notes in Computer Science*, Vol. 1908, pp346-353, Springer Verlag, 2000.
13. Graham E. Fagg, Sathish S. Vadhiyar, Jack J. Dongarra. "ACCT: Automatic Collective Communications Tuning", *Proc of EuroPVM-MPI 2000, Lecture Notes in Computer Science*, Vol. 1908, pp354-361, Springer Verlag, 2000.
14. Sathish S. Vadhiyar, Graham E. Fagg, Jack J. Dongarra. "Automatically Tuned Collective Communications", *Proceedings of SuperComputing2000*, Dallas, Texas, Nov. 2000.