Towards an Accurate Model for Collective Communications*

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 .

1 Introduction

This project developed out of an attempt to build efficient collective communications for a new fault tolerant MPI implementation known as HARNESS [10] FT-MPI [11]. At least 2 different efforts were made 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 [12], [13], we built efficient algorithms for different collective communications and selected the best collective algorithm and segment

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

size for a given {collective communication, number of processors, message 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.

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 actual experiments are 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 Sect.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 Sect.2 reiterate the usefulness of the automatic tuning approach. These results were obtained by conducting the actual experiments with all possible input parameters. In Sect.3, we describe three techniques needed for reducing the large number of actual experiments. In Sect.4, we present some conclusions. Finally in Sect.5, we outline the future direction of the 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.

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.

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

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} 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 system, including clusters of Sparc and Pentium workstations and two different types of PowerPC based IBM SP2 nodes.

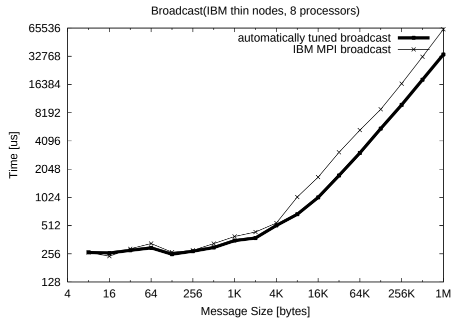
Fig. 1 shows the results for a tuned MPI broadcast on an IBM SP2 using "thin" nodes verses the IBM optimised vendor MPI implementation. Similar encouraging results were obtained for other systems as detailed in [12] & [13].

3 Reducing the Number of Experiments

In the experimental method described in the previous sections a large number of 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

Fig. 1. Broadcast Results (IBM thin nodes)



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.

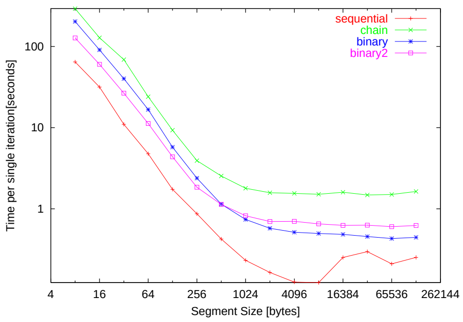
Currently we are examining three techniques to reduce this primary set of experimental points.

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} 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 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. 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.

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



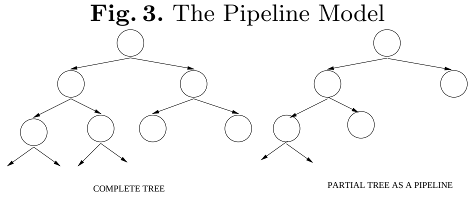
The time taken per iteration for each method is not constant, thus many of the commonly used optimization techniques cannot be used without modification. For example in Fig. 2, a test near the largest segment size is in the order of hundreds of microseconds whereas a single test near the smallest segment 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 show in Fig. 2 is reduced from 12613 seconds to just 39 seconds or a speed up of 318.

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 for the gradient descent methods on the complete topologies or 12613 for the complete exhaustive search.

Fig. 3. The Pipeline Model



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. As part of this approach, we discuss the modeling experiments for broadcast in the following sub sections.

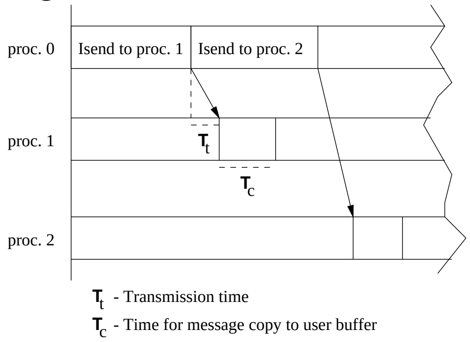
General Overview All the broadcast algorithms are based on a common methodology. The root in the broadcast tree continuously does non-blocking sends of MPI, MPI_Isends, to send individual message buffers to its children. The other nodes post all their non-blocking receives of MPI, MPI_Irecvs, initially.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

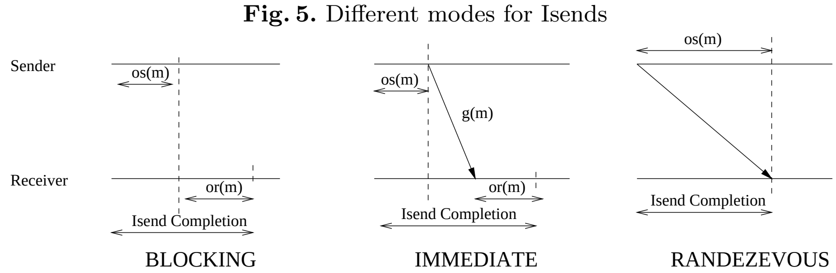
Fig. 4. Illustration of Broadcast Schedule



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 randezevous and are illustrated by Fig. 5

Fig. 5. Different modes for Isends



The parameters associated with the different modes of Isends can vary depending 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).

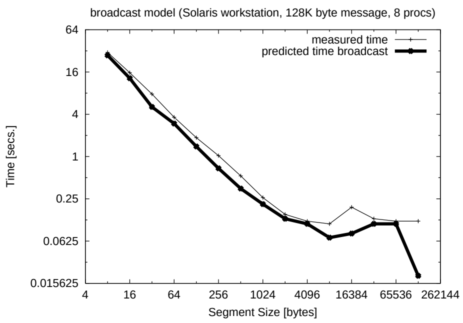
A simple model In this section, we describe a simple model that we have built to calculate the performance of collective communications. At this point, the model is not expected to give good predictions of the performance. A study of the results of this primitive model is useful in understanding the complexities of Isends and developing some insights on building a better model for collective communications.

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.But the model does not use the value of g(m) effectively and it assumes that multiple messages to a node can be sent continuously. The model also does not take into account the number of Isends previously posted.

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 upto 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)+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.

With these simplifying assumptions, the model builds a broadcast schedule for flat, chain, binary and binomial broadcast trees for 2, 4, 8 and 16 processors. Fig. 6 compares the actual and predicted broadcast times for a flat tree broadcast sending a 128K byte message using 8 processors on a Solaris workstation.

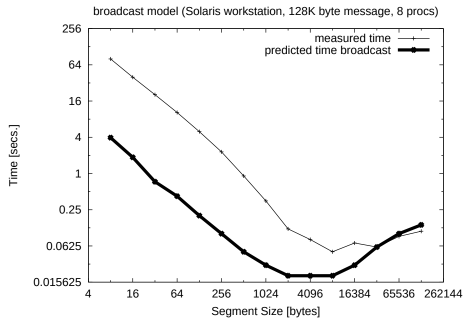
Fig. 6. Flat Tree broadcast



While the model gives good predictions for smaller segment sizes or larger number of segments, it underestimates for smaller number of segments. Also, the performance is poor if the message between the nodes is transmitted as only one segment. For a segment size of 128K, the Isend switches to immediate mode. Since the system has to buffer the messages for immediate Isends, the buffer capacity acts as a bottleneck as the number of posted Isends increase. Since the model does not take into account the number of Isends posted, it gives poor performance for 128K byte messages.

Fig. 7 compares the actual and predicted broadcast times for a chain tree broadcast sending a 128K byte message using 8 processors on the same system.

Fig. 7. Chain tree broadcast



Since the model assumes that messages to a single node can be sent continuously, and since in a chain broadcast tree, the segments are sent continuously to a single node, the model gives much smaller times than the actual times for smaller segment size or for large number of segments.

From the above experiments, we recognize that good models for predicting collective communications have to take into account all the possible scenarios for sends and receives in order to build a good broadcast schedule. While our simplified model did not give good predictions for the results shown, it helped to identify some of the important factors that have to be taken into account for efficient modeling.

4 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.

5 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, Spetmeber 1999. p(469-476).
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* (PpoPP), 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 Synchroniztion. *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 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.
12. 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.
13. Sathish S. Vadhiyar, Graham E. Fagg, Jack J. Dongarra, "Automatically Tuned Collective Communications", *Proceedings of SuperComputing2000*, Dallas, Texas, Nov. 2000.