TOWARDS AN ACCURATE MODEL FOR COLLECTIVE COMMUNICATIONS[1]

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

COMPUTER SCIENCE DEPARTMENT
UNIVERSITY OF TENNESSEE, KNOXVILLE, USA

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.

Key words: MPI, collectives, communications, tuning, model

The International Journal of High Performance Computing Applications,
Volume 17, No. 4, Fall 2003, pp. XX–XX

1 Introduction

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 size} tuple by performing actual experiments with the different algorithms and for different values of message sizes. The tuned collective communication operations were compared with various native vendor MPI implementations. The use of the tuned collective communications resulted in the range of 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 such as 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 because of the exhaustive nature of the testing. 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 three 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 present some conclusions. Finally in Section 5, 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 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 two, 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 1 MB were evaluated.

Phase 3: Repeating phase 1 and phase 2 for different {number of processors, collective operation} combinations. The number of processors will be powers of two 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 being 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.

Figure. 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] and [13].

3 Reducing the Number of Experiments

In the experimental method mentioned in the previous section, about 13,000 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 h of computer time.

Table 1
Collective communication algorithms

Collective communications	Algorithms
Broadcas	Collective communication algorithms
Scatter	Sequential, Chain, Binary and Binomial
Gather	Sequential, Chain, Binary and Binomial
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
Alltoall	Gather followed by scatter, Circular
Barrier	Extended ring, Distributed binomial and tournament

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 64 KB of data, eight 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;
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 three-dimensional (3D) grid of points. For example, varying the message size in powers of two from 8 bytes to 1 MB, processors from 2 to 32 and the MPI operations from Broadcast to All2All, etc.

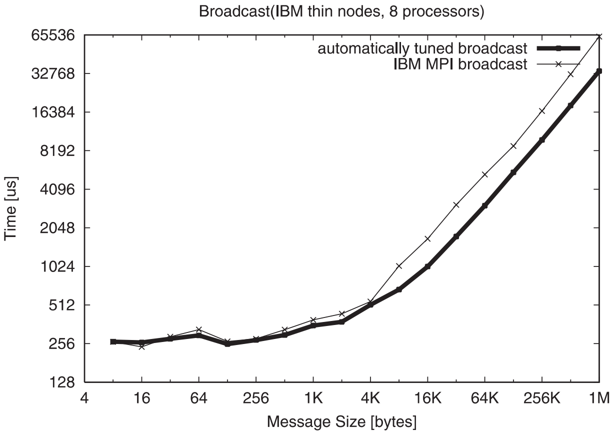

Fig. 1 Broadcast Results (IBM thin nodes).

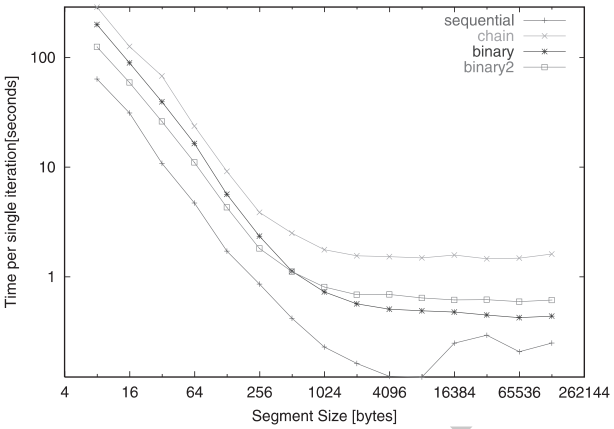

Fig. 2 Segment size versus time for various communication methods.

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 Bytes, 16 Bytes, 32 Bytes, 64 Bytes...1 MB} to {8 Bytes, 1024 Bytes, 8192 Bytes...1 MB}.
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}. Figure. 2 shows the performance of four different methods for solving an eight-processor MPI Scatter of 128 KB of data on a Sparc cluster. Several important points can be observed. First, 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 Figure. 2, a test near the largest segment size is of the order of hundreds of microseconds whereas a single test near the smallest segment size can be of the order of 100 s, 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 that reduce the search space to only the tests close to the optimal values. The first is a modified gradient descent (MGD), and the second is a scanning modified gradient descent (SMGD).

The MGD method is a hill descent (negative gradient hill climber) that searches for the minimum value starting from the largest segment sizes and working in only one direction. This algorithm had to be modified to look beyond the first minimum found so as to avoid multiple local optima.

The SMGD method is a combination of linear search and MGD, where the order of the MGD search is controlled by a sorting of current optimal values and rates of

Table 2
Performance of optimizing algorithms.

Method	Linear time (s)	Linear iteration	MGD time (s)	MGD iteration	SMGD time (s)	SMGD iteration (s)	Speedup
8 proc, 1k bcast	11.4	320	1.3	160	1.3	160	8.8
8 proc, 128K bcast	1324.7	600	21.4	280	10.2	160	130
8 proc, 1k scatter	82.2	320	3.2	160	1.3	100	63
8 proc, 128K scatter	12613.0	600	159.9	220	39.6	90	318

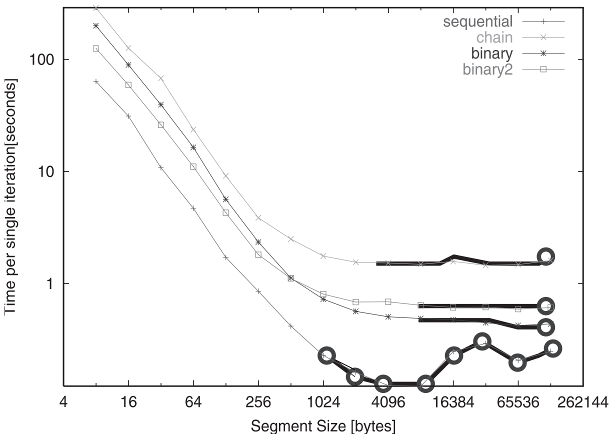

Fig. 3.

change of gradients. The rates of change are used so that we can predict values and thus prune more intelligently. This was required as in many cases the absolute values were insufficient to catch results that interchanged rapidly. This method also includes a simple threshold mechanism that is used to prune the search in cases where a few methods were considerably better than others and thus they can be immediately rejected.

Figure. 3 shows the extent of the MGD and SMGD superimposed on the initial scatter results. The MGD extent is marked by thicker lines and the SMGD by individual points.

Table. 2 lists the relative performance of the algorithms in terms of both experimental time required to find an optimal solution as well as number of iterations. Linear is used to indicate an exhaustive linear search, and speedup is linear compared to the SMGD algorithm. As can be seen from the table, reduction in total time spent finding the optimal can be reduced by a factor of 10 to over 300. Smaller test sets yield less speed up as unnecessary results are less expensive than in larger tests with larger messages.

The number of segment sizes to explore can also be reduced by considering certain characteristics of the

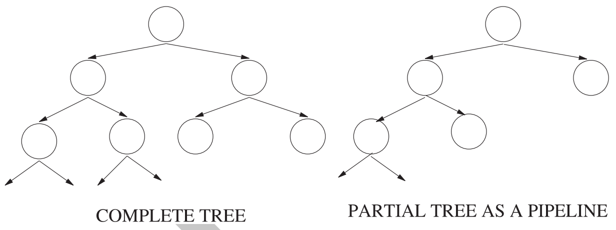

Fig. 4 The pipeline model.

architecture. For example, in some architectures, 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 Figure. 4. In particular, we 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 s per class of communication method (such as tree, sequential or ring). i.e. 6.4 s verses 39 s for the gradient descent methods on the complete topologies or 12,613 s 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

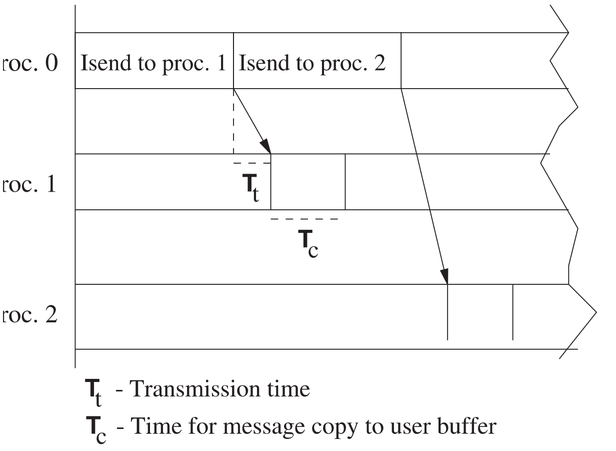
roc. 0
Isend to proc. 1
Isend to proc. 2
roc. 1
T_t
T_c
roc. 2
T_t - Transmission time
T_c - Time for message copy to user buffer

Fig. 5 Illustration of broadcast schedule.

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

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 Figure. 5 can be used to predict the total completion time for the broadcast.

A broadcast schedule such as that shown in Figure. 5 can be used to accurately model the overlap in communications, a feature that was lacking in the parametrized 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*, which were discussed in the parametrized 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 Figure. 6

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-to-point communication times and uses communication schedules for collective operations similar to the broadcast schedule shown in Figure. 5.

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 Figure. 6. 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.

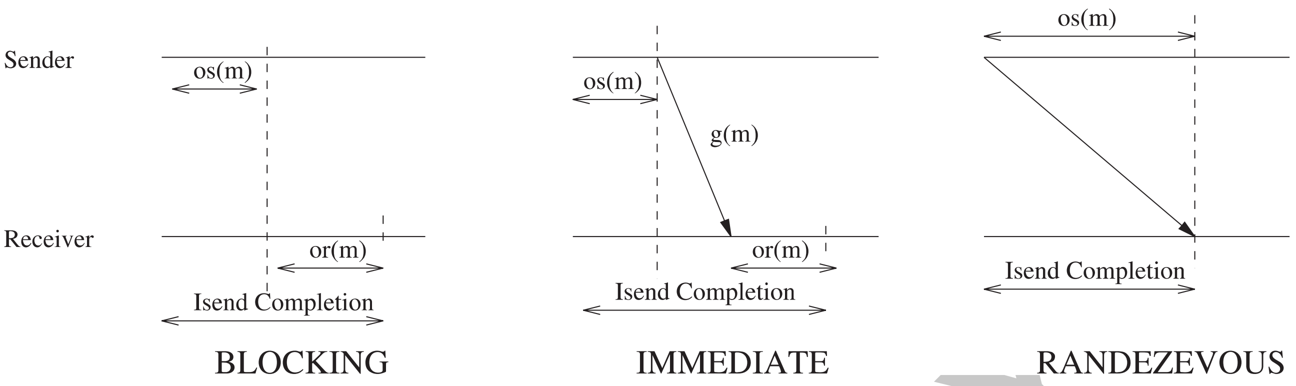

Fig. 6 Different modes for Isends.

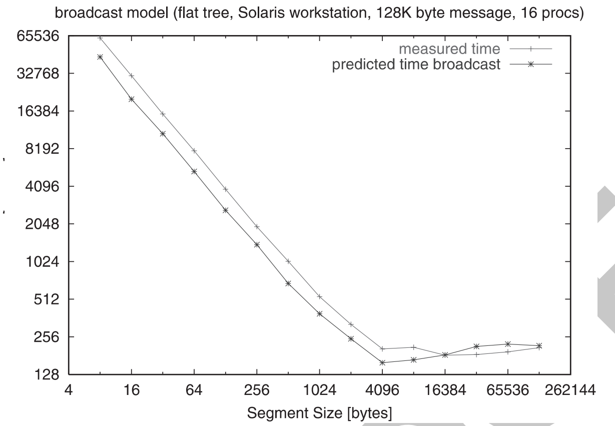

Fig. 7 Flat tree broadcast.

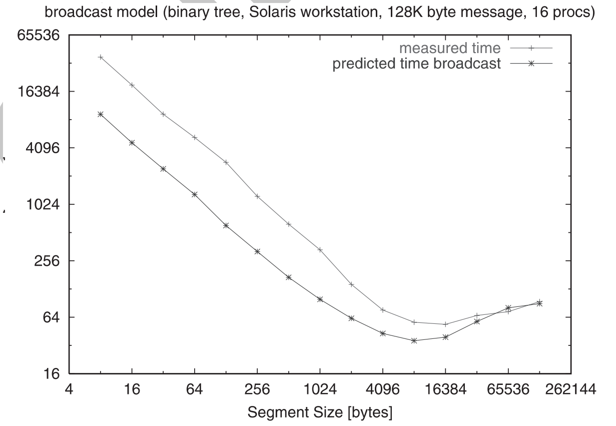

Fig. 8 Binary tree broadcast.

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.

Figure. 7 compares the actual and predicted broadcast times for a flat tree broadcast sending a 128 K byte message using eight processors on a Sparc cluster.

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 128 KB message size is 4 KB, whereas, according to the actual times, the optimum segment size is 16 KB. But the ratio between the actual time at 4 KB and the actual time at 16 KB is found to be just 1.12.

Figure. 8 compares the actual and predicted broadcast times for a binary tree broadcast and Figure. 9 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.

Figure. 10 shows the actual and predicted values of the various broadcast algorithms for 128 K 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

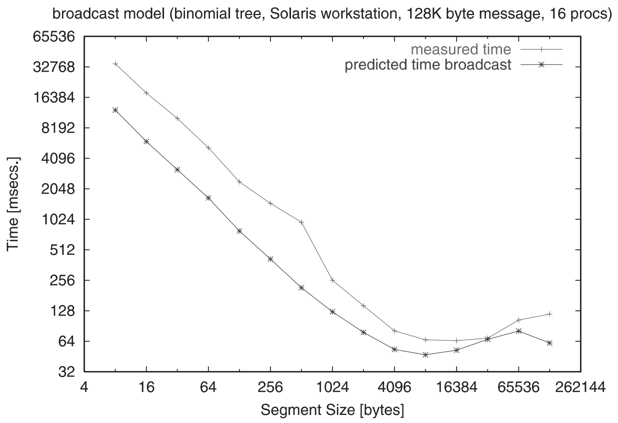

Fig. 9 Binomial tree broadcast.

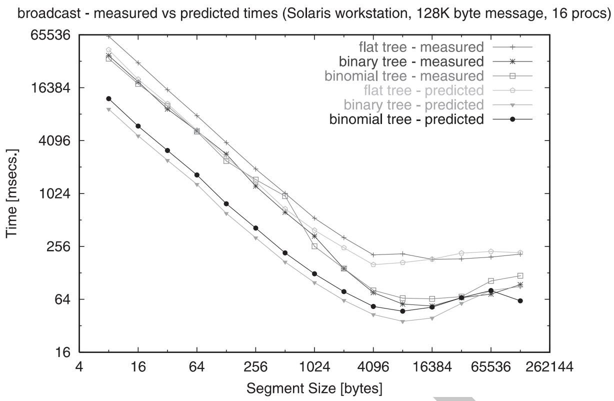

Fig. 10 Comparison of algorithms: measured versus predicted times.

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 such as 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 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 beforehand to determine the point when the performance of Isends degrades and can be plugged into the model.

Models for other collective communications, such as 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 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 is necessary, where various tools for modeling are provided to the user to accurately model the collective communications on his system. 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 with 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.

ACKNOWLEDGMENT

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

BIOGRAPHIES

Sathish Vadhiyar graduated with a Ph.D. in the Computer Science Department at University of Tennessee, USA in May 2003. He did his undergraduate study in the Department of Computer Science and Engineering at Thiagarajar College of Engineering in India in 1997. He obtained his Masters degree in Computer Science at Clemson University, USA in 1999. His research interests are in the area of parallel, distributed and grid computing. His Master's thesis was in the area of parallel compilers. His doctoral research involved building scheduling mechanisms for Grid applications. He is also interested in MPI applications dealing with the issues of building models for MPI collective communications and checkpointing of MPI applications. He is working on adapting numerical libraries to the Grid environments. He was also involved in the multi-institutional Grid project called GrADS. He has published number of papers in the area of parallel, distributed and Grid computing.

Graham Fagg received his B.Sc. in Computer Science and Cybernetics from the University of Reading, UK in 1991 and a Ph.D. in Computer Science in 1998. From 1996 to 2001 he worked as a senior research associate and then a Research Assistant Professor at the University of Tennessee. From 2001 to 2002 he was a visiting guest scientist at the High Performance Computing Center Stuttgart (HLRS). Currently he is a Research Associate Professor at the University of Tennessee. His current research interests include distributed scheduling, resource management, performance prediction, benchmarking, cluster management tools, parallel and distributed IO and high-speed networking. He is currently involved in the development of a number of metacomputing and Grid middle-ware systems including SNIPE/2, MPI_Connect, HARNESS, and a fault tolerant MPI implementation (FT-MPI). He is a member of the IEEE.

Jack Dongarra holds an appointment as University Distinguished Professor of Computer Science in the Computer Science Department at the University of Tennessee and is an Adjunct R&D Participant in the Computer Science and Mathematics Division at Oak Ridge National Laboratory (ORNL) and an Adjunct Professor in Computer Science at Rice University. He specializes in numerical algorithms in linear algebra, parallel computing, and the use of advanced-computer architectures, programming methodology, and tools for parallel computers. His research includes the development, testing and documentation of high-quality mathematical software. He has contributed to the design and implementation of the following open source software packages and systems: EISPACK, LINPACK, the BLAS, LAPACK, ScaLAPACK, Netlib, PVM, MPI, NetSolve, Top500, ATLAS, and PAPI. He has published approximately 200 articles, papers, reports and technical memoranda and he is co-author of several books. He is a Fellow of the AAAS, ACM, and the IEEE and a member of the National Academy of Engineering.

NOTE

1 This work was supported by the US Department of Energy throughcontract number DE-FG02-99ER25378.

REFERENCES

Beck, M., Dongarra, J., Fagg, G., Geist, A., Gray, P., Kohl, J., Migliardi, M., Moore, K., Moore, T., Papadopoulous, P., Scott, S., Sunderam, V. 1999. HARNESS: a next generation distributed virtual machine. *Journal of Future Generation Compute Systems* 15(5–6):571–582.

Culler, D., Karp, R., Patterson, D., Sahay, A., Schauser, K. E., Santos, E., Subramonian, R., and von Eicken, T. May 1993. LogP: towards a realistic model of parallel computation. In *Proceedings of the Symposium on Principles and Practice of Parallel Programming*, San Diego, CA, pp. 1–12.

Fagg, G. E. and Dongarra, J. J. 2000. FT-MPI: fault tolerant MPI, supporting dynamic applications in a dynamic world. In *Proceedings of EuroPVM-MPI 2000*, Lecture Notes in Computer Science Vol. 1908, Springer-Verlag, Berlin, pp. 346–353.

Fagg, G. E., Vadhiyar, S. S., and Dongarra, J. J. 2000. ACCT: automatic collective communications tuning. In *Proceedings of EuroPVM-MPI 2000*, Lecture Notes in Computer Science Vol. 1908, Springer-Verlag, Berlin, pp. 354–361.

Fagg, G. E., Bukovsky, A., and Dongarra, J. J. 2001. HARNESS and fault tolerant MPI. *Journal of Parallel Computing* 27(11):1479–1496.

Frigo, M. 1998. FFTW: an adaptive software architecture for the FFT. In *Proceedings of the ICASSP Conference*, Vol. 3, pp. 1381.

Hensgen, D., Finkel, R., and Manber, U. 1988. Two algorithms for barrier synchronization. *International Journal of Parallel Programming* 17(1):1–17.

Huse, L. P. September 1999. Collective communication on dedicated clusters of workstations. In *Proceedings of the 6th European PVM/MPI Users' Group Meeting*, Barcelona, Spain, LNCS Vol. 1697, Springer-Verlag, Berlin. pp. 469–476.

Kielmann, T., Bal, H. E., and Gorlatch, S. May 2000. Bandwidth-efficient collective communication for clustered wide area systems. In *IPDPS 2000*, Cancun, Mexico.

Prylli, L. and Tourancheau, B. 1998. BIP: a new protocol designed for high performance networking on myrinet. In *PC-NOW Workshop, IPPS/SPDP 1998*, Orlando, FL.

Rabenseifner, R. 1997. A new optimized MPI reduce algorithm.
http://www.hlrs.de/organization/par/services/models/mpi/myreduce.html.

Snir, M., Otto, S., Huss-Lederman, S., Walker, D., and Dongarra, J. 1998. MPI – the complete reference. In *The MPI Core*, Vol. 1, 2nd edition.

Vadhiyar, S. S., Fagg, G. E., and Dongarra, J. J. November 2000. Automatically tuned collective communications. In *Proceedings of SuperComputing2000*, Dallas, TX.

Whaley, R. C. and Dongarra, J. 1998. Automatically tuned linear algebra software. In *SC98: High Performance Networking and Computing*, Orlando, FL. See