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 buffering 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 the modeling of the collective communications based on communication schedules. We compare the accuracy of our models for the broadcast collective communication algorithms with the previously developed MagPIe model. We find that our approach based on communication schedules are more accurate especially for algorithms based on unbalanced tree topologies.

1 Introduction

In our previous work [6, 17], we built efficient algorithms for different MPI [16, 1] 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 approach follows the strategy that is used in efforts like ATLAS [18] for matrix operations and FFTW [7] for Fast Fourier Transforms. 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% to 650% improvement in performance over the native MPI implementations. The tuning system uses point to point communication routines provided by the underlying MPI implementation. The tuned system acts as a separate library and invokes the native MPI functions via the profiling interface. It does not take advantage of any lower-level communications such as hardware-level broadcasting.

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

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 due to 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 phases 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 discuss related work. In Section 3, 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 3 reiterate the usefulness of the automatic tuning approach. These results were obtained by conducting the actual experiments with all possible input parameters. In Section 4, we describe three techniques needed for reducing the large number of actual experiments. In Section 5, we present some conclusions. Finally in Section 6, we outline the future direction of our research.

2 Related Work

There have been number of efforts towards optimizing MPI collective communications. The work by Bruck et. al. [4] optimizes collective communications on LAN networks by providing extensions to operating system kernels. The work by Banikazemi et. al [3] dynamically derives optimal broadcast trees based on the network link parameters. ECO [15] and the work by Karonis et. al [10] derive multi-layer broadcast trees by studying the topology of the underlying networks and are especially suitable for heterogeneous Network of Workstations(HNOWs) and Computation Grids. Most of the above mentioned work does not possess robust models that predict the completion time of the broadcast operation. Our work discusses and validates the simulation models for broadcast.

The work that most closely relates to our effort is MagPIe [14, 11, 12]. MagPIe is intended for optimizing collective communications for clustered wide area systems. The MagPIe models for collective communications are mathematical formulas that predict the cost of collective communications. The mathematical formulas are based on point to point communication parameters measured using the parameterized LogP model [13] which in turn is based on the LogP model [5]. In our work, we identify some of the problems associated with the parameterized LogP model for point to point communications and the MagPIe models for collective communications. We propose efficient substitutes for both the models and compare the efficiency of our models with the MagPIe approach.

3 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.

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 [2]
Allreduce	Reduce followed by broadcast, Allgather followed by operation, Chain, Binary, Binomial and Rabenseifner [2]
Allgather	Gather followed by broadcast
Alltoall	Gather followed by scatter, Circular
Barrier	Extended ring, Distributed binomial and tournament [8]

Table 1. Collective communication algorithms

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.

3.1 Results For Broadcast

The experiments consist of multiple 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 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 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.

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. The tuned broadcast results are compared with the results of the broadcast using IBM optimized vendor MPI implementation. Similar encouraging results were obtained for other systems as detailed in [12] and [13].

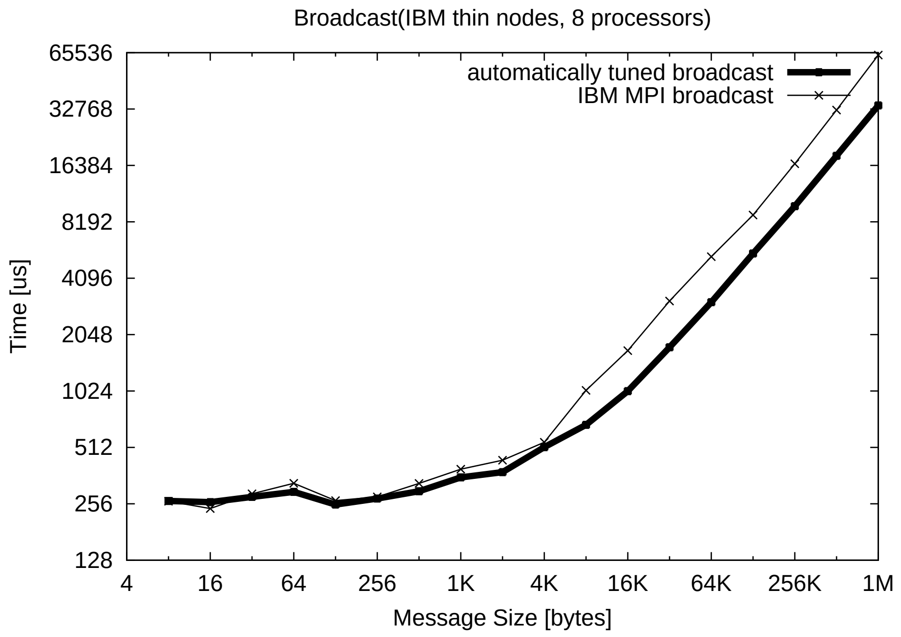

Fig. 1. Broadcast Results (IBM thin nodes)

The times for broadcast shown in Figure 1 and the various figures in the other sections of the paper were obtained by calling the broadcast functions within a loop of fixed number of iterations and placing time stamps before and after the loop. Barrier calls are also placed after the broadcast calls to ensure that all the processes begin the broadcast operation at the same time. The difference between the time stamps are calculated and the cost of the barrier functions are also taken into account. The resulting time is divided by the number of iterations to obtain the time for the broadcast function. The entire procedure is illustrated by the code section shown in Figure 2.

```
/* Time for a broadcast function */

  MPI_Barrier(comm);

  if(rank == 0){
  /* first processor */
    gettimeofday(&t1, NULL);
  }

  for(i=0; i<loop_count; i++){
    broadcast_function(); /* invocation of broadcast function */
    MPI_Barrier(comm);
  }
  if(rank == 0){
    gettimeofday(&t2, NULL);
  }

  if(rank == 0){
    diff = time_diff(t1, t2); /* difference between the time stamps */
    total_time = diff - (loop_count-1)*barrier_cost;
    total_time /= loop_count;

  }
```

Fig. 2. Timing Procedure

4 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 of computer time.

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, 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.

These techniques for reducing the time for conducting actual experiments are described in the following sections. The experiments corresponding to the results shown in the following sections of the paper were conducted on 3 different parallel environments detailed as follows:

1. 31 node Sun Microsystems SunBlade 100 System with 500 Mhz UltraSPARC-IIe processors and 512 MBytes of Memory per node. System located at the Department of Computer Science, University of Tennessee. Interconnection 100 Mbps 100-Base-T Ethernet interface, full duplex and connected to Cisco switches 2948 running Solaris 2.8 and MPICH 1.2.3.
2. 40 thin node IBM SP2 with 160MHz POWER2 SC processors and 256 MBytes of memory per node located at the Joint Institute for Computational Science, University of Tennessee. Interconnection TB3 Switch and adapter with peak bandwidth of 150 MB/sec running AIX Version 2, Release 4 and native IBM MPI.
3. 512 node Cray T3E-900/512 with 128 MBytes of memory per node. Located at the High Performance Computing Center (HLRS), Stuttgart, Germany. Interconnect 3D torus interconnect with 500MB/Sec bidirectional transfers per link. System running unicosmk 2.0.5.55 and the mpt 1.4.0.2 toolkit.

4.1 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 MB, 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 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.

4.2 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 3 shows the performance of four different methods for solving an 8 processor MPI Scatter of 128 KB of data on the UltraSPARC cluster. Both the x-axis and the y-axis data are logarithmically scaled. 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 Figure 3, 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 that reduce the search space to only the tests close to the optimal values. The first is a Modified Gradient Decent (MGD), and the second is a Scanning Modified Gradient Decent (SMGD).

The MGD method is a hill decent (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 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

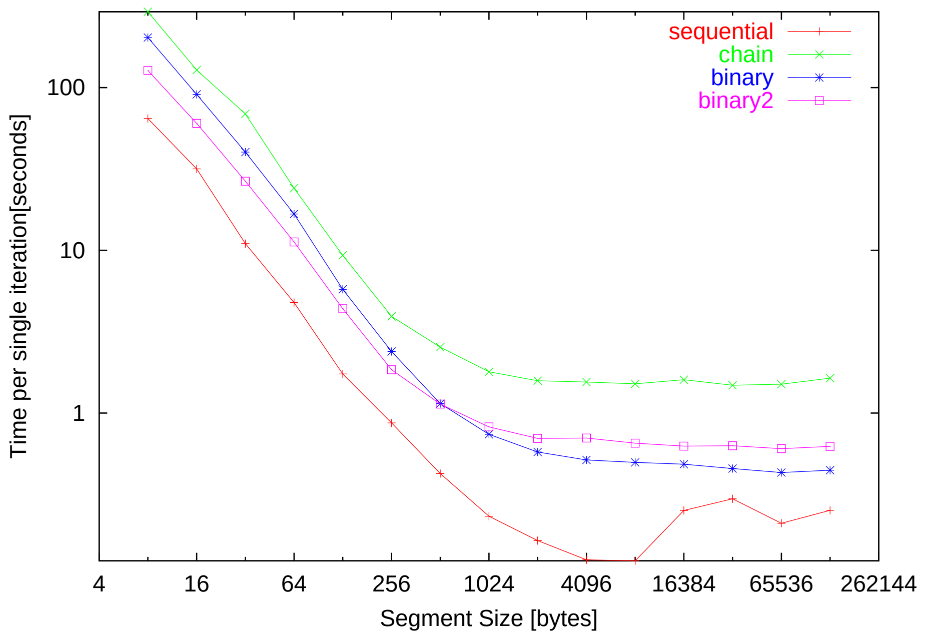

Fig. 3. Segment size vs. time for various communication methods

and thus they can be immediately rejected. It should be noted that the SMGD method checks all algorithms with the maximum segment size always.

Figure 4 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. On the SMGD search some of the methods results are so comparatively large compared to others that they are pruned immediately. The pruning performed by the MGD is much less aggressive as can be seen by the extent of the thicker lines. In the case of methods that do not exhibit the shown curves characteristics the MGD can provide more accurate but slower results.

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 speed up 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 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.

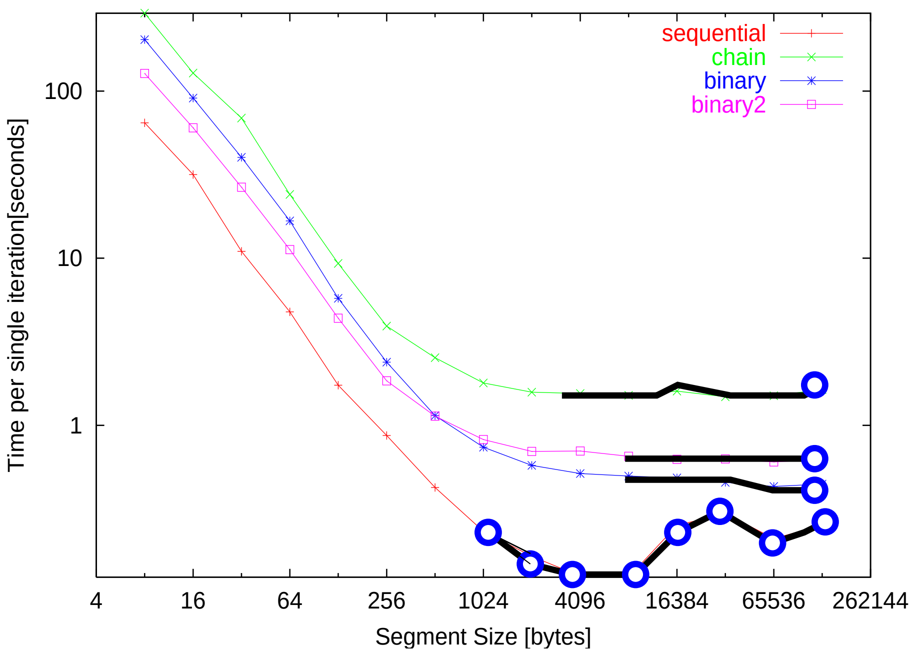

Fig. 4. Gradient Descent Methods for Reducing Search Space

Method	*Linear Time (seconds)*	*Linear Iteration*	*MGD Time (seconds)*	*MGD Iteration*	*SMGD Time (seconds)*	*SMGD Iteration (seconds)*	*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

Table 2. Performance of optimizing algorithms

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 5. 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.

Although the pipeline test is useful, it cannot always be applied as it assumes that the network interconnection is capable of full cross-sectional bandwidth. In the case of many large clusters this is not true. For intermediate cases we have devised a further test to check for network saturation based on pair wise communications so that we can decide if we can use this optimization atall.

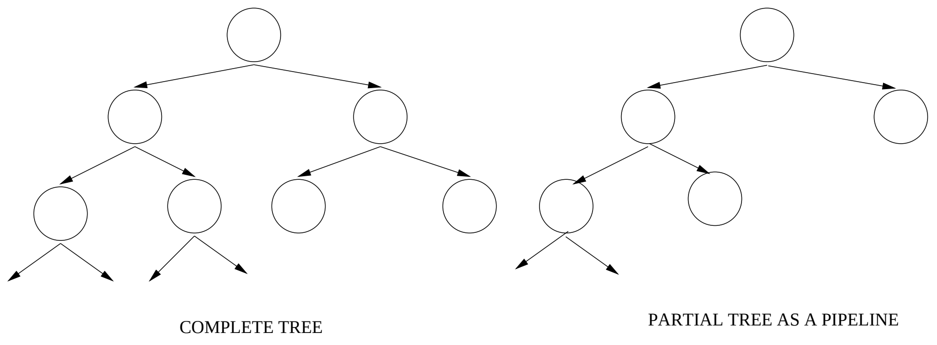

Fig. 5. The Pipeline Model

4.3 Reducing the single-experiment time

Running the actual experiments to determine the optimal 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 the raw hardware characteristics such as the link bandwidth, latency, topology etc. Instead they take into account times for MPI point to point communications in terms of 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 performs MPI non-blocking sends (MPI_Isends), to send individual message buffers to its children. The other nodes post all their MPI non-blocking receives (MPI_Irecvs), initially. The nodes between the root node and the leaf nodes in the broadcast tree, send on any segments to their children as soon as any segments are received.

After determining the times for individual Isends and the times for message receptions, a broadcast schedule as illustrated by Figure 6 can be used to predict the total completion time for the broadcast.

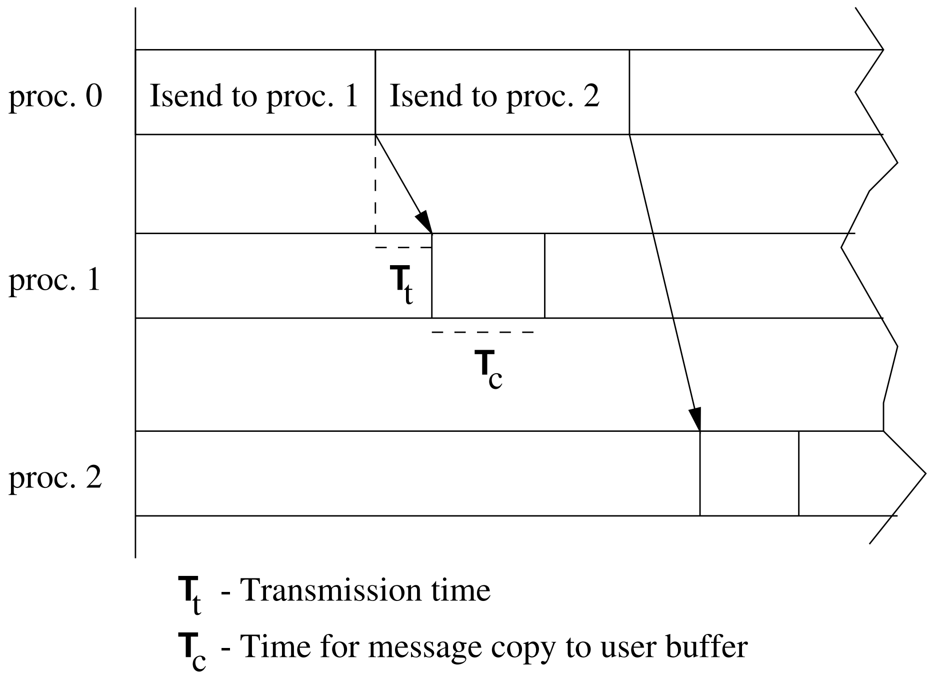
proc. 0
Isend to proc. 1
Isend to proc. 2
proc. 1
T_t
T_c
proc. 2
T_t - Transmission time
T_c - Time for message copy to user buffer

Fig. 6. Illustration of Broadcast Schedule

A broadcast schedule such as the one shown in Figure 6 can be used to provide better predictions for collective communication algorithms based on unbalanced tree topologies than the prediction models based on mathematical formulas such as in MagPIe [11, 12, 14].

Measurement of Point to Point Communication Parameters As observed in the previous section, accurate measurements of the times for Isends and the times for the reception of the messages are necessary for efficient modeling of broadcast operations. Previous communications models [9, 14, 11, 12] 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 [13] based on LogP model [5] 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 7.

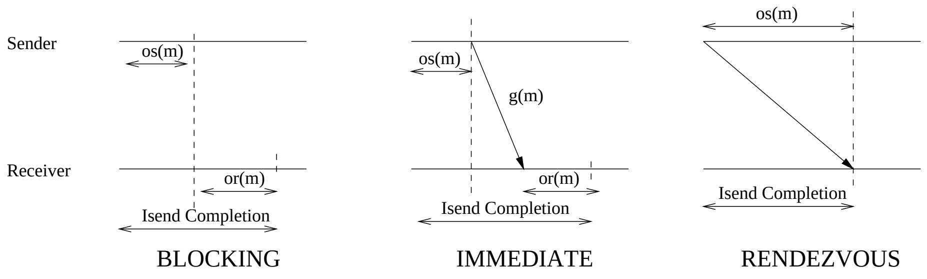

Fig. 7. Different modes for Isends

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). To illustrate the dependence of the Isends on the number of previous Isends, we conducted an experiment on the UltraSPARC cluster in which one process of a MPI program continuously posts Isends of messages and another process of the program continuously receives the messages using Irecvs. Figure 8 shows the times taken for the individual Isends for messages of sizes 2K and 4K bytes.

In Figure 8, we observe that the overhead of Isend is not constant for all the Isends. This is due to the limitations on the capacities of the buffers used in the underlying network layer. For messages of small size, most MPI implementations copy the messages to the local buffers before sending the messages to the receiver. When the buffers become fully occupied, a posted Isend waits till some of the messages in the buffers are cleared. For the UltraSPARC MPICH implementation, we found the buffer capacities to be 32 KBytes. Hence in Figure 8, we observe steep increase in Isend times for the 16th and 8th Isends corresponding to the 2K and 4K messages respectively.

To obtain the various parameters corresponding to point to point communications, we first measure *os_g_or(m)* for message size, m. *os_g_or(m)* is the

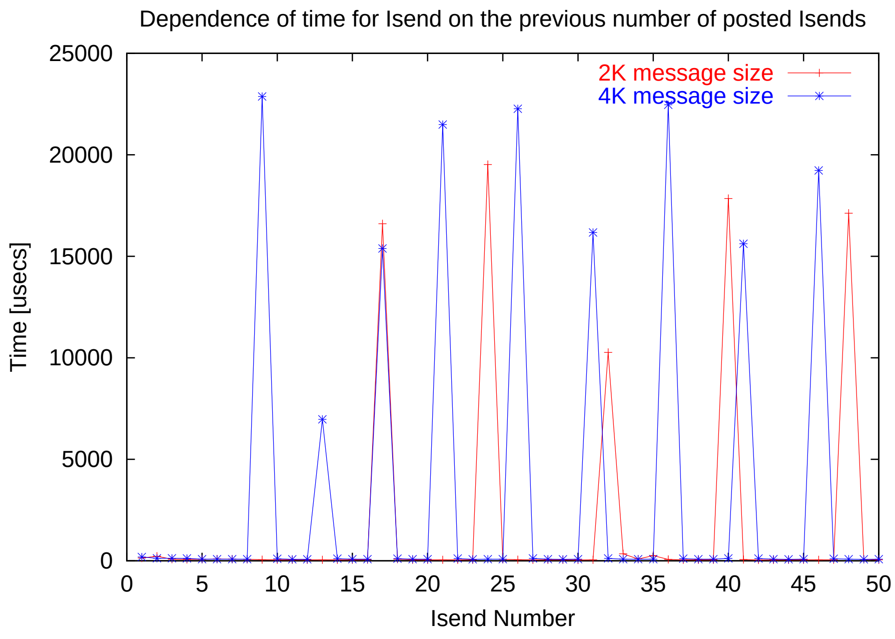

Fig. 8. Time for Isends on the UltraSPARC cluster

time from the start of the Isend on the sender to the time taken for the Irecv to complete on the receiver and is obtained by ping-pong transmission of the message using MPI_Isends, MPI_Irecvs and MPI_Waits. We then measure the receive overhead, *or(m)*, by forcing the receiver to wait for *os_g_or(m)* before receiving a message from the sender. If the resulting *or(m)* is comparable to the *os_g_or(m)*, then the underlying MPI implementation used blocking or rendezvous mode for Isend of the message. If *or(m)* is less than *os_g_or(m)*, the underlying MPI implementation used the immediate mode of Isend. In this case, we measure *os_start(m)*, the time for Isend when the underlying network buffers possess adequate capacity to accommodate messages. We calculate the gap, *g(m)* as $g(m) = os\_g\_or(m) - (os\_start(m) + or(m)$.

We then proceed to determine the different parameters associated with overheads corresponding to Isends. As noted in Figure 8, the times for Isends for small messages are initially small and approximately equal to *os_start(m)*. After *initial_interval(m)* number of Isends, the Isends follow a pattern of an Isend of large overhead, *os_spike(m)*, followed by *mid_interval(m)* number of Isends of small overheads, *os_mid(m)*, followed again by an Isend of large overhead, *os_spike(m)* and so on. We calculate the various parameters of the Isend overhead by having the sender send consecutive 16 messages, 32 messages etc. to the receiver till 10 Isends of large overheads are encountered. We then calculate the average of large overheads as *os_spike(m)*, the initial number of Isends of small overheads as *initial_interval(m)*, the average number of Isends between two successive Isends of large overheads as *mid_interval(m)* and the average of times of the Isends corresponding to the *mid_interval(m)* as *os_mid(m)*. Thus by associating 5 different parameters corresponding to the send overhead, viz., *initial_interval(m)*, *os_start(m)*, *os_spike(m)*, *mid_interval(m)* and *os_mid(m)*, better predictions of the point to point communication behavior can be obtained than when using a single parameter, *os(m)*, as is being practiced in LogP [5] and parameterized LogP [13] models.

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 microseconds.

Model Based on Communication Schedules Predictions of broadcast algorithms were achieved by building communication schedules for the algorithms as shown in Figure 6 and using the various point to point communication parameters previously calculated. The procedure for prediction is the simulation of a parallel broadcast algorithm by a sequential program. The simulator maintains events corresponding to the various sends and receives and calculates the starting and ending times for the sends and receives.

One of the important utilities the simulator uses to simulate the Isends is the *isend()* utility. The *isend()* utility accepts as input, *pending_g_count* and returns as output the Isend time for message size, *m*. The *pending_g_count* is the number of incomplete *g(m)*s corresponding to the previous Isends. Based on the pattern of Isend overheads determined from the point to point communication measurements and the number of pending *g(m)*s, the *isend()* utility determines the time for Isend to be either *os_start(m)*, *os_spike(m)* or *os_mid(m)*. The simulator, before invoking the *isend()* utility, determines *pending_g_count* as

$$pending\_g\_count = \frac{previous\_Isend\_count \times g(m) - time\_since\_first\_Isend}{g(m)} \tag{1}$$

The simulation of the send and receive events for broadcast operation is illustrated by means of the pseudo code shown in Figure 9.

Experiments and Results Experiments were conducted on the UltraSPARC, IBM SP and Cray T3E machines to verify the accuracy of our prediction models. We also compared the prediction results from our method, the Automatically Tuned Collective Communications (ATCC), with the prediction results from MagPIe.

Figures 10 - 13 show the results obtained on the Cray T3E-900 for various broadcast algorithms. The figures show the actual and predicted times for broadcast for different segment sizes when the message size used was 256 KBytes. The predicted times were obtained for both ATCC and MagPIe models.

```
recv start time:

  if(intermediate node)
    recv_time_time = last_Isend_end_time;
  else
    if(first recv)
      recv_start_time = 0;
    else
      recv_time_time = last_recv_end_time;
    end
  end

recv end time:

  if(blocking or rendezvous mode)
    recv_end_time = max(recv_start_time,
                        corresponding_Isend_start_time) + os_g_or(m);
  else if(immediate mode)
    if(recv_start_time <= corresponding_Isend_end_time)
      recv_end_time = corresponding_Isend_start_time + os_g_or(m);
    else
      if(first recv)
        recv_end_time = corresponding_Isend_end_time +
                              g(m) + or(m);
      else
        recv_end_time = max(precv_recv_end_time - or(m),
                        corresponding_Isend_end_time) + g(m) + or(m);
      end
    end
  end

send start time

  if(root)
    send_start_time = prev_Isend_end_time;
  else
    send_start_time = max(prev_Isend_end_time,
                         corresponding_recv_end_time;
  end

send end time

  if(blocking or rendezvous)
    send_end_time = send_start_time + os_g_or(m);
  else
    calculate pending_g_count;
    isend_time = isend_time(pending_g_count);
    send_end_time = send_start_time + isend_time;
  end
```

Fig. 9. Pseudo code for the simulation of send and receive events for broadcast operation

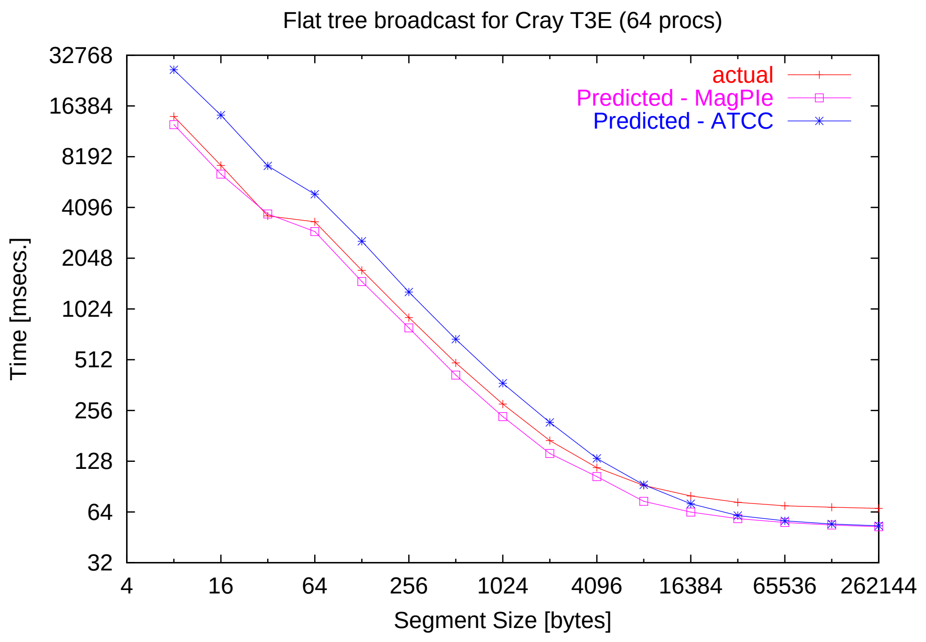

Fig. 10. Flat Tree broadcast on Cray T3E

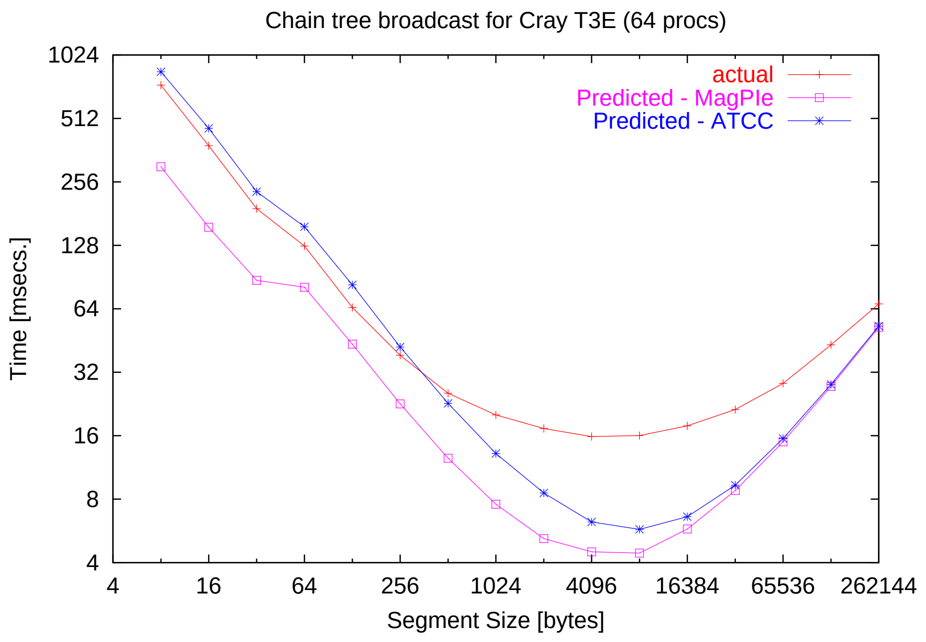

Fig. 11. Chain Tree broadcast on Cray T3E

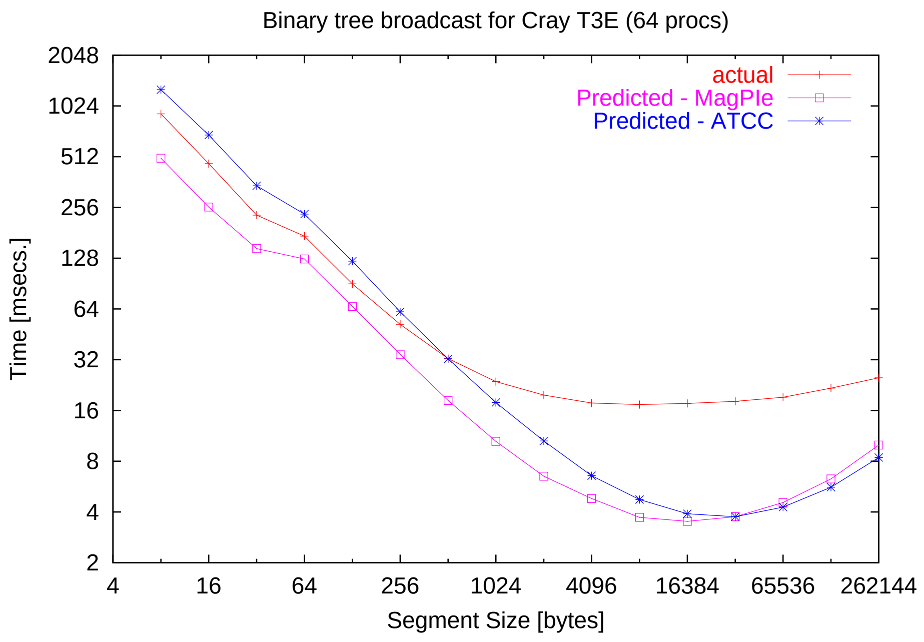

Fig. 12. Binary Tree broadcast on Cray T3E

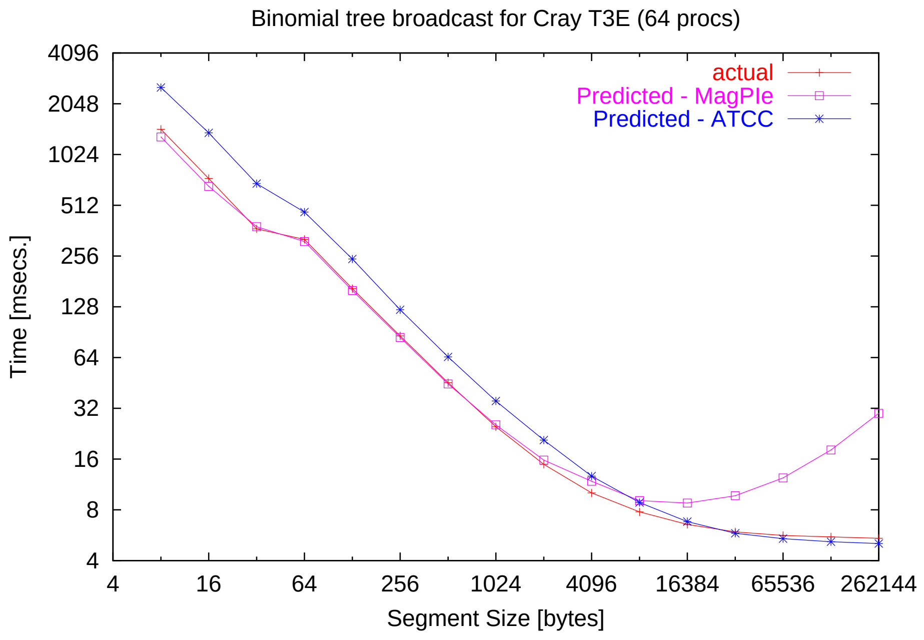

Fig. 13. Binomial Tree broadcast on Cray T3E

The actual optimum segment sizes and the optimum segment sizes as determined by ATCC and MagPIe are given in Table 3. The nearness of the predicted optimal values to the actual optimal values are also given in the table.

Broadcast Topology	*Actual Optimum Segment Size*	*MagPIe Optimum Segment Size*	*ATCC Optimum Segment Size*	*(Actual Time at MagPIe Optimum) / (Actual Time at actual Optimum)*	*(Actual Time at ATCC Optimum) / (Actual Time at actual Optimum)*
Flat	256 KB	256 KB	256 KB	1.00	1.00
Chain	4 KB	8 KB	8 KB	1.01	1.01
Binary	8 KB	16 KB	32 KB	1.02	1.05
Binomial	256 KB	16 KB	256 KB	1.28	1.00

Table 3. Prediction Accuracy on a T3E-900

From Figures 10 - 13 and Table 3, we find that both MagPIe and ATCC provide the same kind of predictions for the broadcast times and the near-optimal segment sizes for flat, chain and binary trees. For binomial trees, ATCC provides much better prediction than MagPIe especially for large segment sizes. This is due to the unbalanced nature of the binomial tree. The upper bounds of the broadcast times as predicted by the mathematical formulas of MagPIe are much higher than the actual times taken for broadcast for large segment sizes. In these cases, ATCC provides a more accurate prediction due to the employment of communication-schedule based simulations.

Similar results were obtained on the Sun UltraSPARC and the IBM SP2 system. The predictions for the unbalanced binomial tree algorithms on the Sun UltraSPARC and the IBM SP2 system for 256 KByte message sizes are given in Figures 14 and 15.

Figures 16 - 18 compares the various broadcast algorithms in terms of the actual measurements, MagPIe predictions and ATCC predictions respectively on the Cray T3E set of 64 processors. The figures show the times for various segment sizes when message size of 256 KByte was used.

From the figures, according to the actual measurements, flat tree broadcast algorithm gives the worst performance for all segment sizes and the MagPIe and ATCC predictions concur with the actual measurements for flat tree broadcast. According to the actual measurements, the chain broadcast algorithm provides best performance for segment sizes 8bytes to 1Kbytes. The binomial broadcast algorithm provides best performance for segment sizes greater than 32 KBytes. According to MagPIe and ATCC predictions, the chain broadcast algorithm provides best performance for segments sizes 8 bytes to 4 KBytes. MagPIe predicts that binary tree algorithm provides the best performance for segment sizes 4KBytes to 256 KBytes. ATCC predicts that the binary tree algorithm provides the best performance for segment sizes 4KByes to 64 KBytes and the binomial

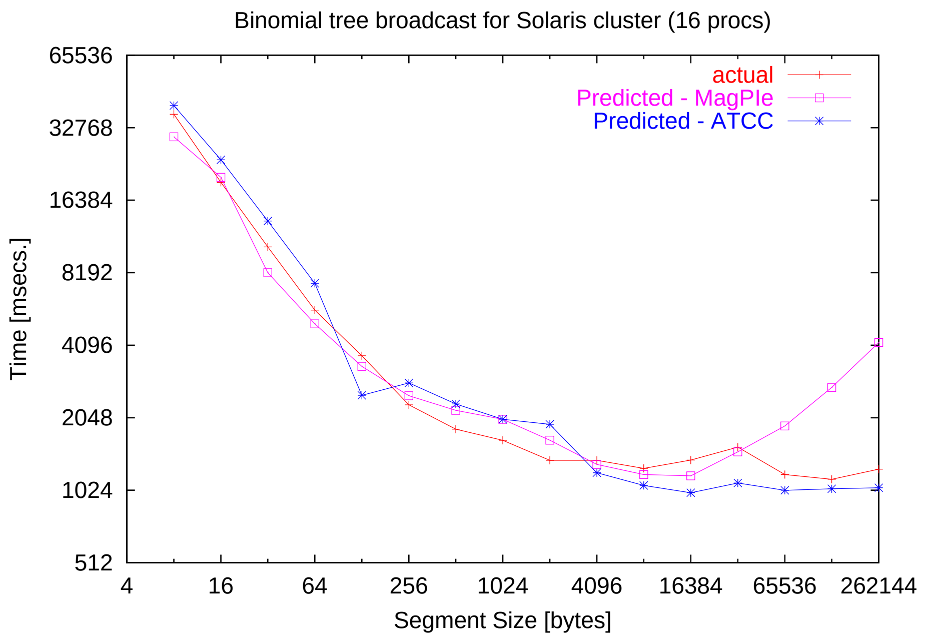

Fig. 14. Binomial Tree broadcast on Sun UltraSPARC

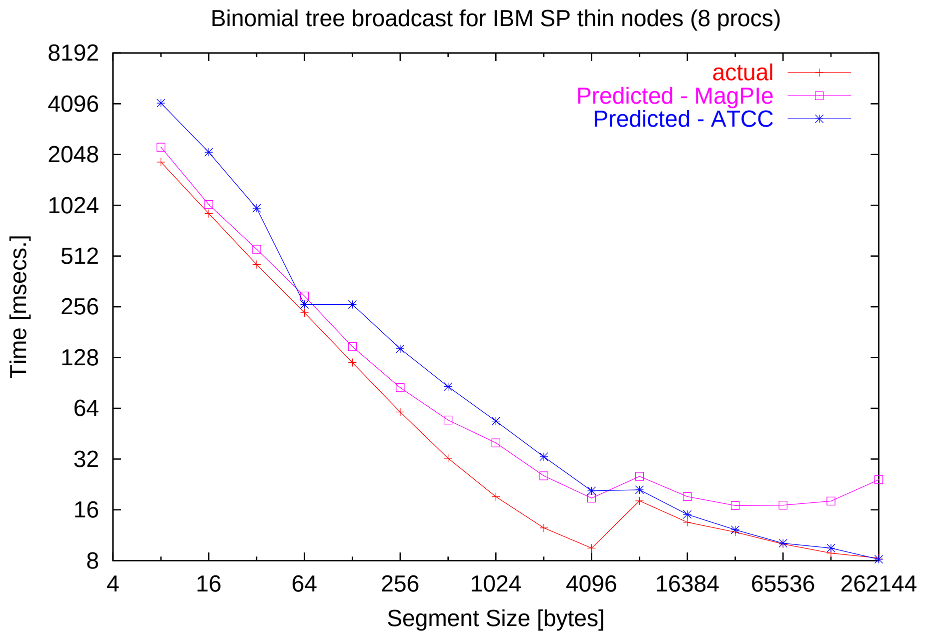

Fig. 15. Binomial Tree broadcast on IBM SP2

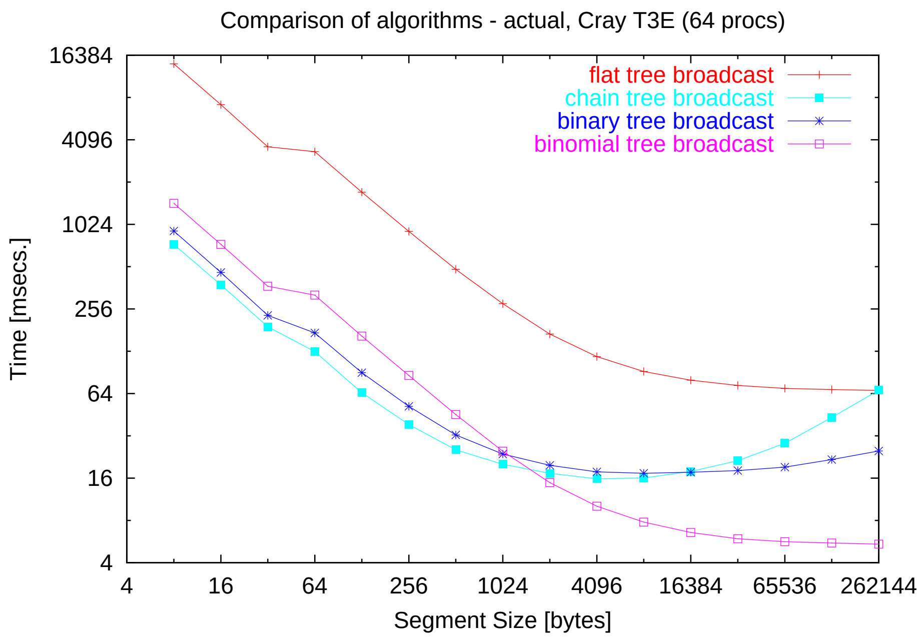

Fig. 16. Comparison of broadcast algorithms on Cray T3E - actual measurements

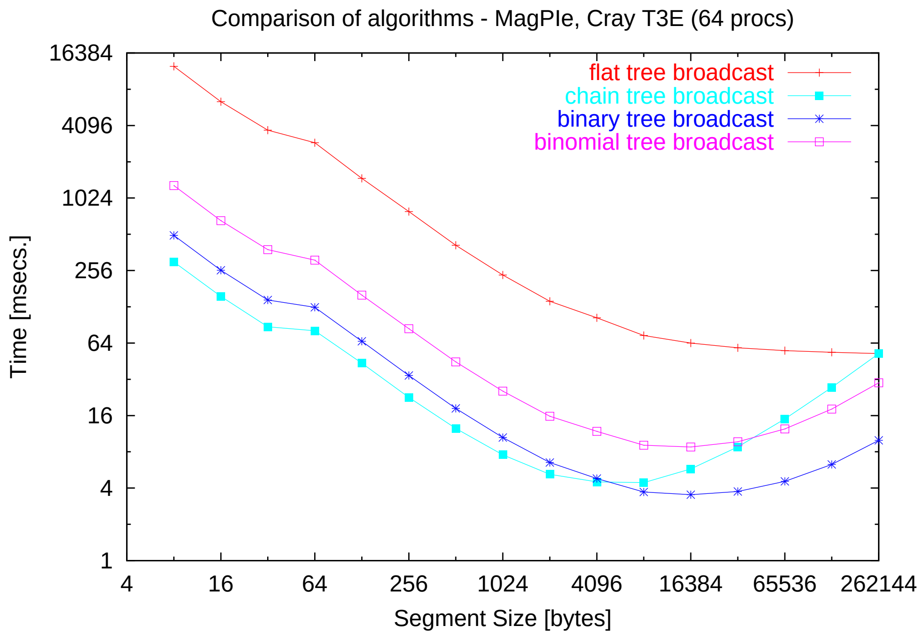

Fig. 17. Comparison of broadcast algorithms on Cray T3E - MagPIe predictions

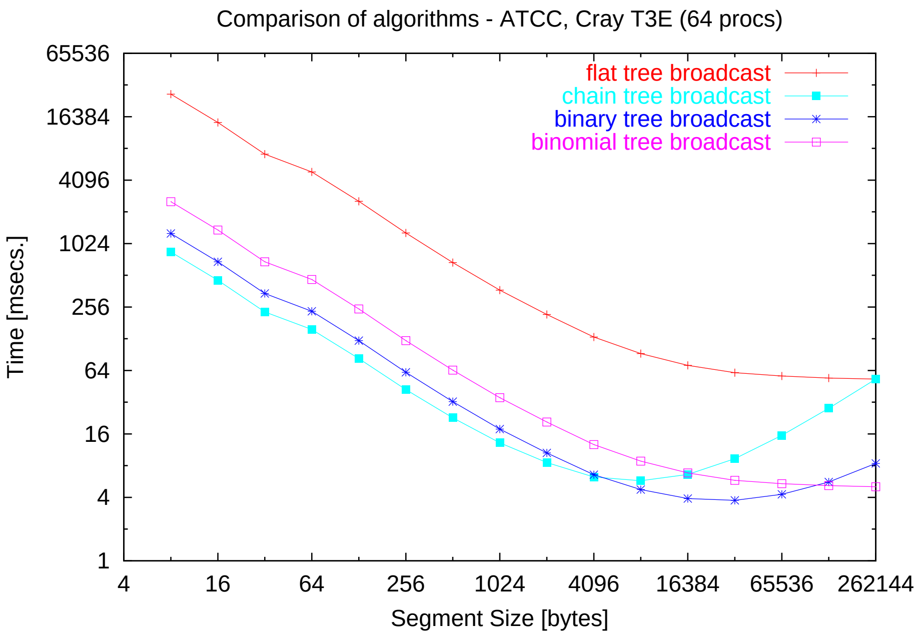

Fig. 18. Comparison of broadcast algorithms on Cray T3E - ATCC predictions

tree algorithm provides the best performance for segment sizes 128 and 256 KBytes. Also, MagPIe predicts that the chain and the binomial tree algorithms provide almost the same performance for large size segments, whereas according to the actual measurements and the ATCC predictions, there is a significant difference in performance between chain and binomial broadcast algorithms for large segment sizes. Thus for medium segment sizes, both MagPIe and ATCC provide the same amount of accuracy in comparing the performance of the various broadcast algorithms. For large segment sizes, ATCC prediction concur more with the actual results in the comparison of the different broadcast algorithms.

Thus we find that the ATCC model is able to predict both near-optimal segment sizes within a single algorithm and near-optimal algorithms when comparing different algorithms.

Modeling Other Collective Communications 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 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 this can be plugged directly into our 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.

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 various collective communications. While proving the efficiency of the techniques, we have also compared our model with a well known model for collective communications and found some encouraging results.

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. We also want to investigate the efficacy of using the parameters of point to point communications corresponding to the previous models, especially LogP and parameterized LogP, in our communication-based schedules.

7 Acknowledgments

The authors would like to thank Michael Resch and Thomas Beisel of the High Performance Computing Center (HLRS), Stuttgart, Germany for allowing us to conduct our experiments on the centers Cray T3E machine.

References

1. MPI. *http://www-unix.mcs.anl.gov/mpi.*
2. A New Optimized MPI Reduce Algorithm. *http://www.hlrs.de/structure/support/parallel_computing/models/mpi/myreduce.html*, 1997.
3. M. Banikazemi, V. Moorthy, and D. K. Panda. Efficient Collective Communication on Heterogeneous Networks of Workstations. In *ICPP*, pages 460–467, 1998.
4. J. Bruck, D. Dolev, C. Ho, M.C. Rosu, and H. R. Strong. Efficient Message Passing Interface (MPI) for Parallel Computing on Clusters of Workstation. *Journal of Parallel and Distributed Computing*, 40(1):19–34, 1997.
5. D. 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 *Proceedings of the Symposium on Principles and Practice of Parallel Programming*, pages 1–12, San Diego, CA, May 1993.
6. G. Fagg, S. Vadhiyar, and J. Dongarra. ACCT: Automatic Collective Communications Tuning. In *Proceedings of EuroPVM-MPI 2000, Lecture notes in Computer Science, Vol. 1908*, pages 354–361. Springer Verlag, 2000.
7. M. Frigo. FFTW: An Adaptive Software Architecture for the Fft. In *Proceedings of the ICASSP Conference*, volume 3, page 1381, 1998.
8. D. Hensgen, R. Finkel, and U. Manber. Two Algorithms for Barrier Synchronization. *International Journal of Parallel Programming*, 17(1), 1998.
9. L. P. Huse. Collective Communication on Dedicated Clusters of Workstations. In *Proceedings of the 6th European PVM/MPI Users' Group Meeting*, volume Lecture Notes in Computer Science, Vol. 1697, pages 469–476, Barcelona, Spain, September 1999.
10. N. Karonis, B. de Supinski, I. Foster, W. Gropp, E. Lusk, and J. Bresnaha. Exploiting Hierarchy in Parallel Computer Networks to Optimize Collective Operation Performance. In *In Proceedings of the Fourteenth International Parallel and Distributed Processing Symposium (IPDPS '00)*, pages 377–384, Cancun, Mexico, May 2000.
11. T. Kielmann, H. E. Bal, and S. Gorlatch. Bandwidth-efficient Collective Communication for Clustered Wide Area Systems. In *International Parallel and Distributed Processing Symposium (IPDPS 2000)*, pages 492–499, Cancun, Mexico, May 2000.

12. T. Kielmann, H. E. Bal, S. Gorlatch, K. Verstoep, and R. F. H. Hofman. Network Performance-aware Collective Communication for Clustered Wide-area Systems. *Parallel Computing*, 27(11):1431–1436, 2001.
13. T. Kielmann, H. E. Bal, and K. Verstoep. Fast Measurement of LogP Parameters for Message Passing Platforms. In *Lecture Notes in Computer Science, Vol. 1800*, pages 1176–1183, 4th Workshop on Runtime Systems for Parallel Programming (RTSPP) held in conjunction with IPDPS 2000, Cancun, Mexico, May 2000.
14. T. Kielmann, R. F.H. Hofman, A. Plaat H. E. Bal, and R. A.F. Bhoedjang. MagPIe: MPI's Collective Communication Operations for Clustered Wide Area Systems. In *Proceedings of Seventh ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP'99)*, pages 130–140, Atlanta, GA, May 1999.
15. B. Lowekamp and A. Beguelin. ECO: Efficient Collective Operations for Communication on Heterogeneous Networks. In *In Proceedings of the 10th International Parallel Processing Symposium*, pages 399–405, Honolulu, Hawaii, April 1996.
16. M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra. *MPI: The Complete Reference - The MPI Core*, volume 1. Boston MIT Press, 2nd edition, September 1998.
17. S. Vadhiyar, G. Fagg, and J. Dongarra. Automatically Tuned Collective Communications. In *Proceedings of SuperComputing2000*, November 2000.
18. R. C. Whaley and J. Dongarra. Automatically Tuned Linear Algebra Software. In *SC98: High Performance Networking and Computing*, 1998.