Automatically Tuned Collective Communications[*][*]

Sathish S. Vadhiyar[+], Graham E. Fagg[*], Jack Dongarra
Computer Science Department
University of Tennessee, Knoxville
Presenting author[+]: vss@cs.utk.edu
Correspondence author[*]: fagg@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. In this paper, we discuss an approach in which the collective communications are tuned for a given system by conducting a series of experiments on the system. We also discuss a dynamic topology method that uses the tuned static topology shape, but re-orders the logical addresses to compensate for changing run time variations. A series of experiments were conducted comparing our tuned collective communication operations to various native vendor MPI implementations. The use of the tuned collective communications resulted in about 30%-650% improvement in performance over the native MPI implelementations.

1. INTRODUCTION

This project developed out of an attempt to build efficient collective communications for a new fault tolerant MPI implementation known as HARNESS FT-MPI [10]. 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 parametrized LogP model. Mathematical models based on few network parameters in the system do not adequately take into account the overlap in communication that occurs in collective communications.

In this paper, we discuss an approach in which the optimum algorithm and optimum buffer size for a given collective communication on a system is determined by conducting experiments on the system. This approach follows the strategy that is used in efforts like ATLAS [7] for matrix operations and FFTW [6] for Fast Fourier Transforms. The experiments were conducted in several phases. In the first phase, the best buffer size for a given algorithm for a given number of processors is determined by evaluating the performance of the algorithm for different buffer sizes. In the second phase, the best algorithm for a given message size is chosen by repeating the first phase with a known set of algorithms and choosing the algorithm that gives the best result. In the third phase, the first and second phase are repeated for different number of processors.

The large number of buffer sizes and the large number of processors significantly increase the time for conducting the above experiments. Modified hill-descent heuristics to reduce the number of experiments to find the optimal buffer size and optimal algorithm for a given message size for a given collective communication on a given number of processors are developed and tested. These heuristics reduce the number of experiments by factors of 10-100.

In Section 2, we examine the different algorithms that are available in our repertoire. In Section 3, we describe the machines we used, the experiments conducted on the machines, and analysis of the results. In Section 4, we describe the hill descent heuristics that are used in reducing the number of experiments to find the oprimal parameters. In Section 5, we discuss the dynamic topology method that reorders the processes within a given toplogy for communication. In Section 6, we present some conclusions. Finally in Section 7, we outline the future direction of the research.

2. ALGORITHMS FOR COLLECTIVE COMMUNICATIONS

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

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

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

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

2.1 Sequential tree

In this topology, the root sends the messages successively to all the other processors. If there are n processors, this algorithm takes n-1 steps to complete. Since the latencies are not chained, this algorithm gives good performance in wide-area networks.

2.2 Chain and Ring trees

In the chain and ring trees, the root sends to process 1 and process N-1 receives from N-2. Process r $\in$ [1. . . N-2] receives from r-1 and sends to r+1. Though process N must wait for N-1 time steps for the reception of the message, the pipelined nature of the algorithm gives successive operations high throughput.

2.3 Binary tree

Each node but the root receives from one node, and all sends to up to two other nodes. This algorithm takes about O($\log_2$N) steps to complete.

2.4 Binomial tree

The definition of the binomial tree as given in the paper by Laurs Paul Huse is "In s $\in$ [1... ln] steps, process 0 in all groups send to $r_x = \lfloor (2 + maxr)/2 \rfloor$ which receive from 0. All groups with more than two processes are then split in $\phi = [0...r_x - 1]$ and $\psi = [r_x...maxr]$ and new ranks $r' = r - r_x$ assigned to ψ."

In most cases, the binomial tree algorithm gives better performance than the binary tree.

2.5 Rabenseifner's Algorithm

Rabenseifner [4] has implemented a version of reduce and allreduce which complete in O(N)+O(count) time where N is the number of processors and count is the message size. Due to the distribution of computation on all the nodes, the bandwidth in the algorithm is about 2.7 times better than the bandwidth in the binary algorithm. But the latency is worse. Hence the algorithm must be used if the message size is bigger than a particular limit. This limit varies from one system to another and has to be found by conducting experiments on the system.

2.6 Circular Algorithm for Alltoall

In step s $\in$ [1 ...N], each process r sends to (r+s)modN and receives from (r-s+N)modN.

2.7 Distributed binomial for barrier

In step s $\in$ [1... ln], process r sends to (r+s)mod N and receives from (r-s)mod N. At the end of ln steps, each process is in synchronization with every other process.

2.8 Extended Ring for Barrier

This is similar to the ring algorithm except that after the completion of a single ring, in which the root receives the message from process N-1, another communication sequence is initiated in which the root resends the message to process 1, process 1 to process 2 and so on till the message is finally received by process N-2.

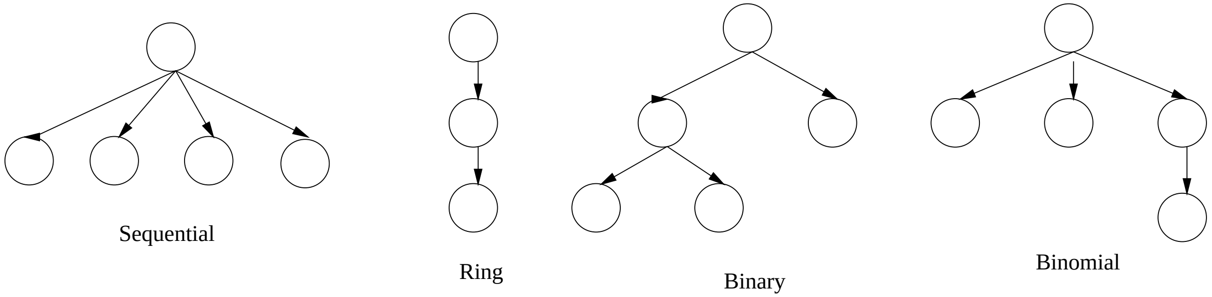
2.9 Tournament Algorithm for Barrier

In this algorithm[9], barrier is viewed as playing a tournament in which a process that recieves (wins a game) continues to wait for the next message (moves to the next game of the tournament) while a process that sends (loses a game) exits out of contention. At the end of $\lceil \log_2 N \rceil$ steps, process 0 broadcasts a message to the other processes.

3. EXPERIMENTAL SETUP AND RESULTS

The experiments consist of many phases. In the first phase, we determine the best segment size for a given message size for a given algorithm for a collective operation. The segment sizes are powers of 2, multiples of the basic data type and less than the message size. Having conducted the first phase for all the algorithms, we determine the best algorithm for a collective operation for a given message size. Message sizes from the size of the basic data type to 1MB were evaluated. This forms the second phase of the experiments. Though we have conducted the experiments on only 8 processors, the third phase of the experiments would be to evaluate the results on a set of different number of processors. 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.

Figure 1: Different topologies for communications



The experiments were conducted on 4 different systems. 2 of the systems are named Cetus and Torc, available in our department.

The Cetus system is composed of 31 workstations connected by 100 Mbps switched Ethernet. Each workstation is a 143-MHz UltraSPARC processor, with 256-Mbytes memory, 16 KB on-chip instruction and data cache, 512 KB external cache, 100 Mbps 100-Base-T Ethernet interface and 2.1-Gbyte internal fast SCSI-2 disk running Solaris 2.5. The Torc system is a collection of dual processor (300/450 MHz) and single processor (450/600 MHz) Linux/NT machines connected by 100Mbit Ethernet, Giganet and Myrinet interconnections.

Figure 2: Broadcast Results(cetus)

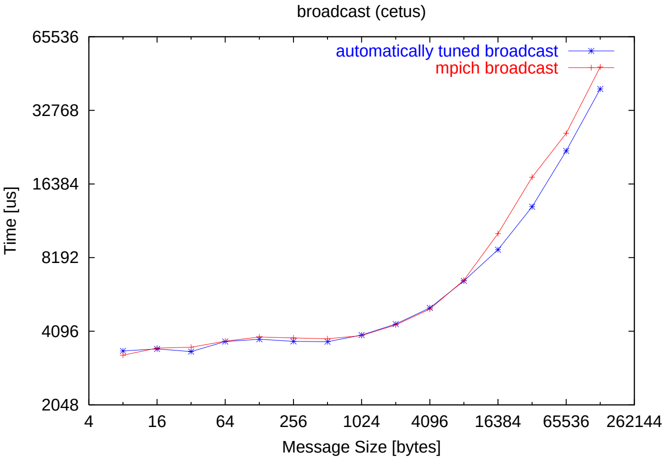


Figure 2 compares the performance of the algorithms for broadcast with the mpich1.1.2 algorithm on Cetus machines with 8 processors. Table 2 shows the optimal algorithm and the optimal buffer size chosen for the Cetus system. Since the Cetus machines are workstations connected by Ethernet links, the optimal algorithm chosen is the binomial algorithm. The mpich's algorithm for broadcast is a type of binomial algorithm and hence follows closely with the performance of our algorithms up to message size of 8K. For message sizes larger than 8K, the underlying network sends messages in packets of 8K. Hence algorithms that segment message into 8K packets perform better than mpich algorithm that sends the entire message. Figure 3 shows the prformance of different broadcast algorithms on cetus machines and illustrates that selection of certain algorithms like *pure* and *chain* can result in degradation of performance over mpich broadcast.

Table 2: Broadcast Results(cetus)

Message Size (bytes)	*Optimal Algorithm*	*Optimal Buffer Size (bytes)*
8	binomial	8
16	binomial	16
32	binary	32
64	binomial	64
128	binomial	128
256	binomial	256
512	binomial	512
1K	binomial	1K
2K	binomial	2K
4K	binomial	4K
8K	binomial	4K
16K	binomial	8K
32K	binomial	8K
64K	binomial	8K
128K	binomial	8K

Figure 3: Broadcast Result(cetus)

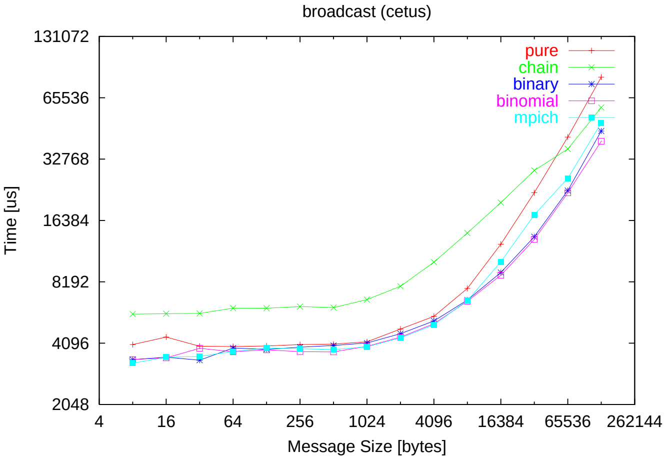


Figure 4 and Table 3 show the results on Torc machines with 8 processors running Linux and interconnected by 100Mbs Ethernet. Because of the Fast Ethernet link, the overhead associated with the communication dominates the *gap* times [3] for the network on Torc. Since in binary and chain algorithms, a processor communicates with only few other processors, these algorithms are able to utilize the gap values more efficiently than the other algorithms. Hence these algorithms combined with message segmenting help in improving the performance over mpich's algorithm . The mpich's binomial algorithm does not give a good performance on Torc since a processor does not immediately send the next segment of a message to another processor as soon as the first segment is sent.

Figure 4: Broadcast Results(torc)

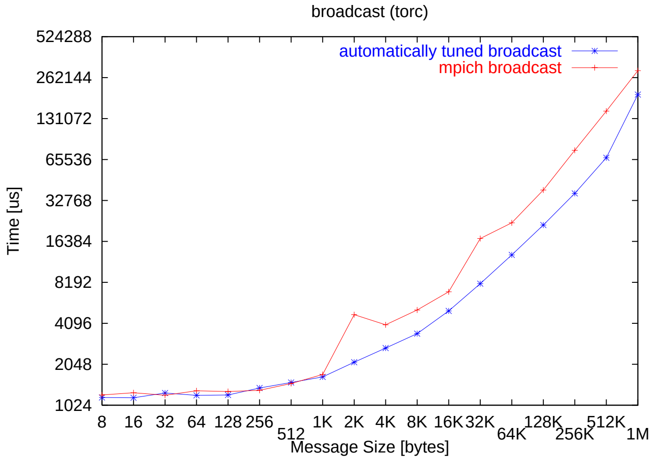


Table 3: Broadcast Results(torc)

Message Size (bytes)	*Optimal Algorithm*	*Optimal Buffer Size (bytes)*
8	binomial	8
16	binomial	16
32	binary	32
64	binomial	64
128	binomial	128
256	binomial	256
512	binomial	512
1K	sequential	1K
2K	binary	2K
4K	binary	2K
8K	binary	2K
16K	binary	4K
32K	binary	4K
64K	chain	4K
128K	chain	4K
256K	chain	4K
512K	chain	4K
1M	binary	4K

Experiments were also conducted on the IBM SP2 system available in JICS (Joint Institute of Computational Science). The high performance switch has 150 MB/s peak bandwidth. The SP2 has 34 nodes: 2 high nodes and 32 thin nodes running AIX 4.2. Their configurations are:

High Nodes: POWERPC 604 8-way chips, running at 112 MHz, with 1 GByte of shared memory.

Old Thin Nodes: POWER2 SC chips, running at 120 MHz, with 256 MBytes of memory and 2.2 GB disk drives each.

New Thin Nodes: POWER2 SC chips, running at 160 MHz, with 256 MBytes of memory and 4.4 GB disk drives each.

The MPI collective algorithms were implemented on top of IBM MPI, IBM's implementation of MPI. The performance of the collective algorithms was evaluated on the old thin nodes using 8 processors and on a single high node using all the 8 processors. The performance was compared with that of IBM MPI.

Figure 5 and Table 4 show the results on old thin nodes.

Figure 5: Broadcast Results (IBM thin nodes)

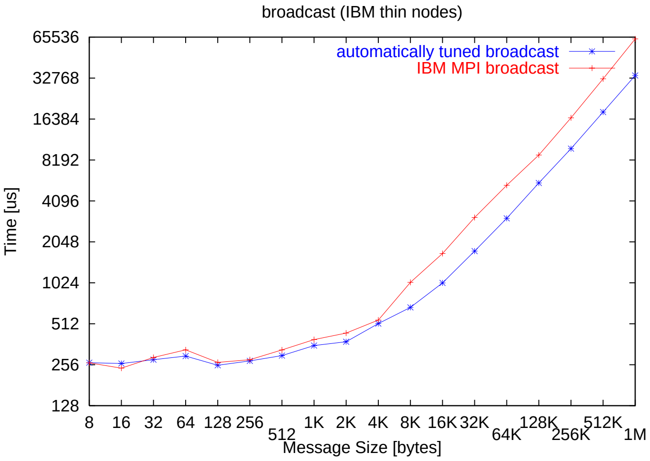


Table 4: Broadcast Results(IBM thin nodes)

Message Size (bytes)	*Optimal Algorithm*	*Optimal Buffer Size (bytes)*
8	sequential	8
16	sequential	16
32	sequential	32
64	binomial	64
128	sequential	128
256	sequential	256
512	binomial	512
1K	sequential	1K
2K	sequential	2K
4K	binomial	2K
8K	binary	4K
16K	binary	4K
32K	binomial	4K
64K	binomial	4K
128K	chain	4K
256K	chain	4K
512K	chain	4K
1M	chain	32

The superior performance of the communication adapter results in very small gap values. Hence the binary and the chain algorithms combined with message segmentation give better performance than the IBM MPI algorithm for message sizes larger than 8K bytes.

Figure 6 and Table 5 show the results on high node 8-way SMPs. IBM MPI sends and receives take place through the communication adapter. This results in large gap values for communication between nodes on a SMP. These gap values are utilized by the overlap in communication in binomial algorithms. This results in superior performance over IBM MPI which tries to use the same algorithm for communication on both thin and high nodes. Thus different algorithms have to be used on the same system for different memory models.

Figure 6: Broadcast Results (IBM high nodes)

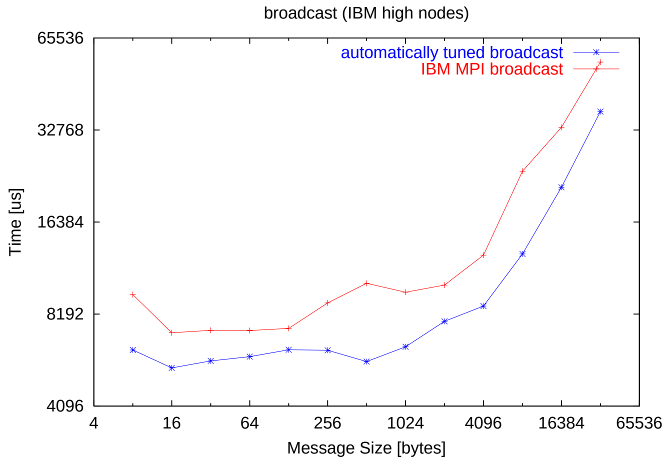


Table 5: Broadcast Results(IBM high nodes)

Message Size (bytes)	*Optimal Algorithm*	*Optimal Buffer Size (bytes)*
8	binomial	8
16	binomial	16
32	binomial	32
64	binomial	64
128	binomial	128
256	binomial	256
512	binomial	512
1K	binomial	1K
2K	binomial	2K
4K	binomial	4K
8K	binomial	4K
16K	binomial	4K
32K	binomial	4K

Figures 7- 12 and Tables 6- 9 show the results for other collective communications.

4. REDUCING THE NUMBER OF EXPERIMENTS

In the experimental method described in the previous sections a large number of individual experiments have to be

Figure 7: Scatter

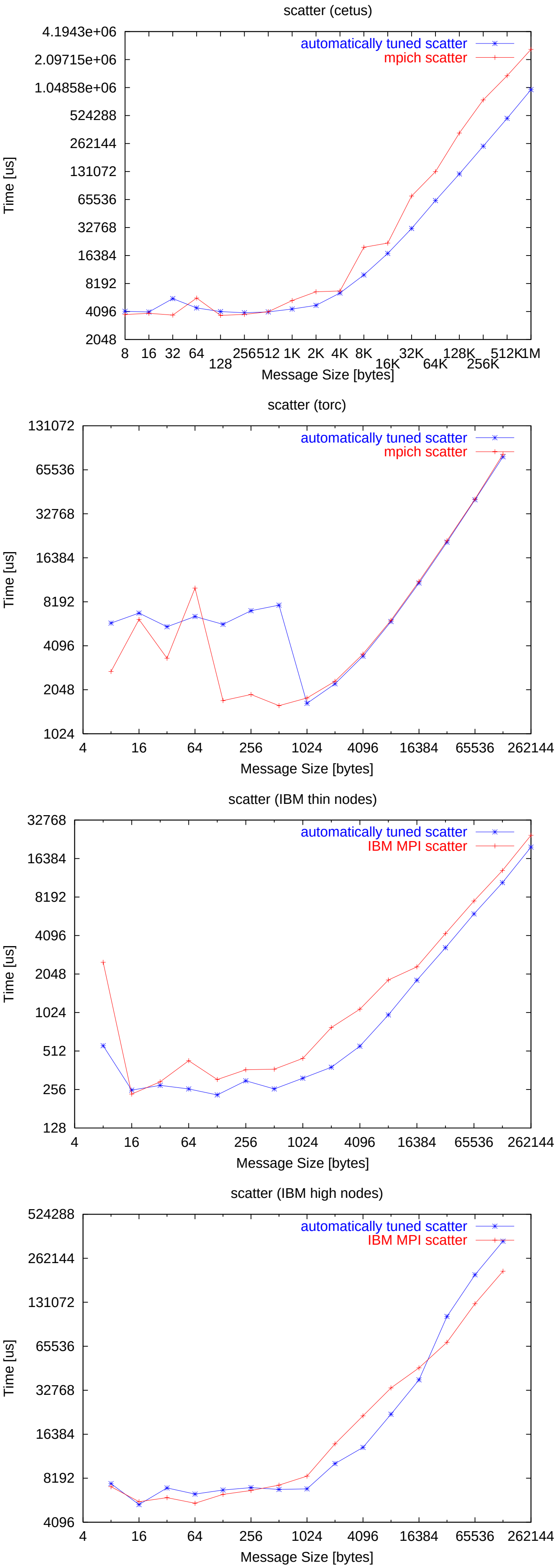


Figure 8: Gather

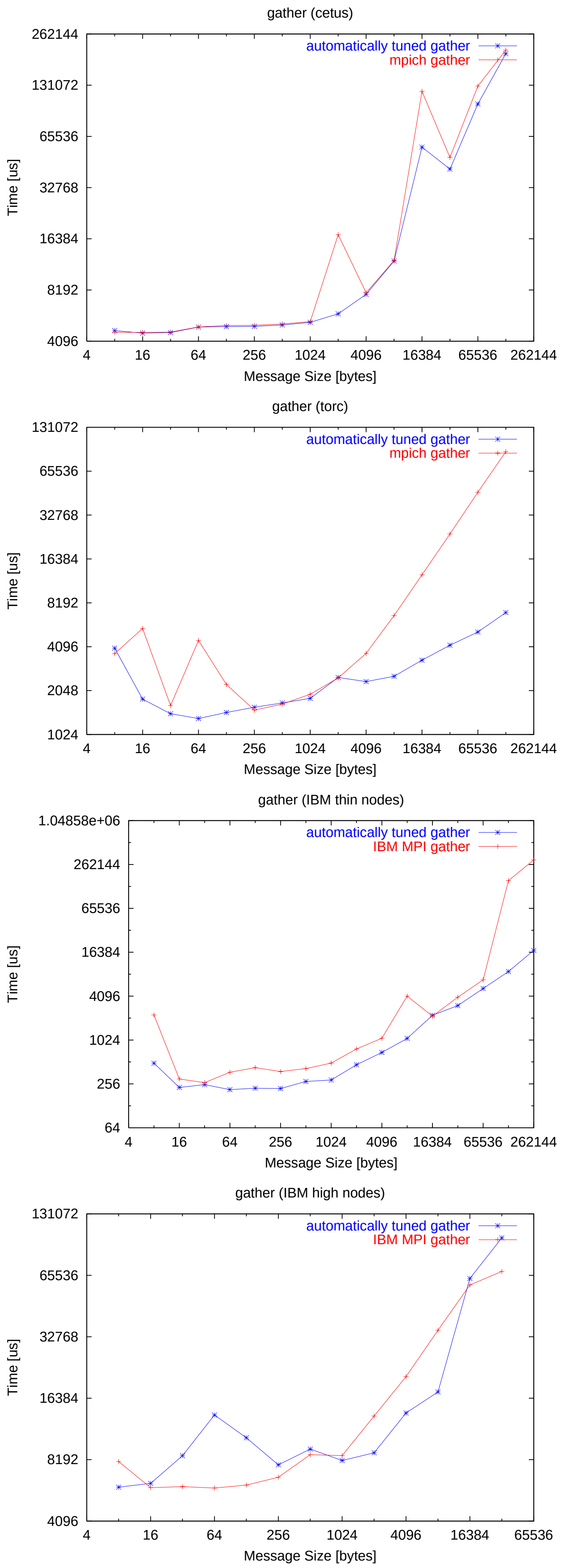


Figure 9: Reduce

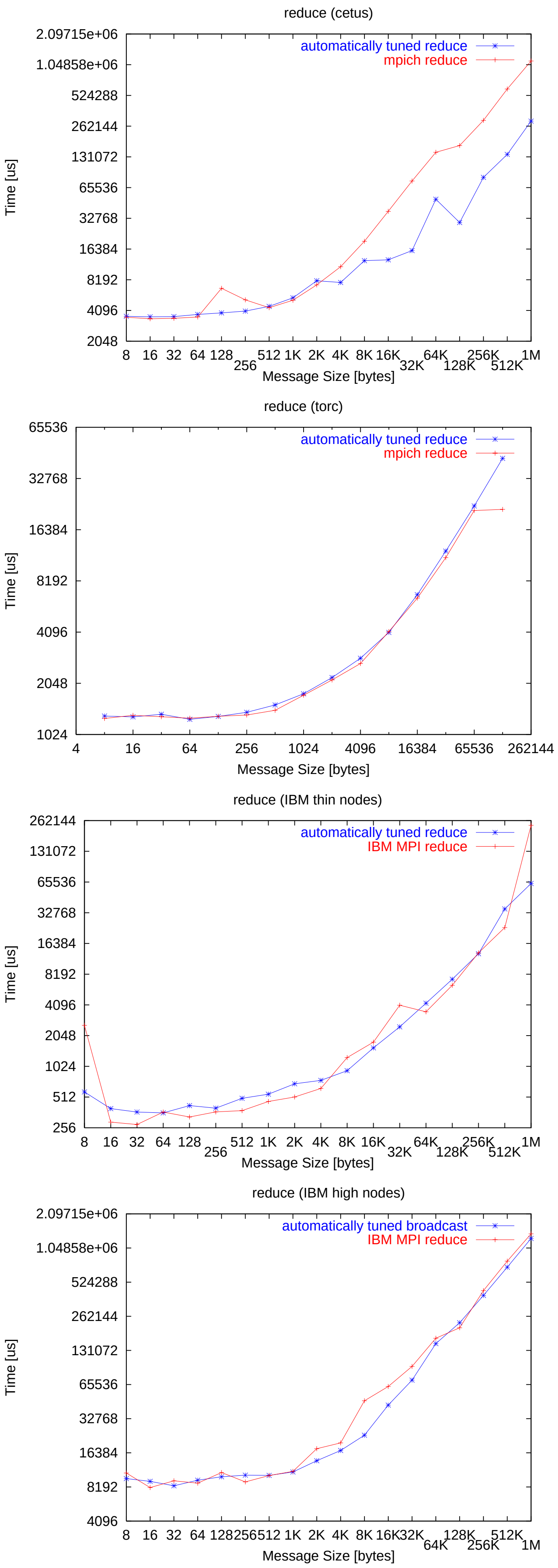


Figure 10: Allreduce

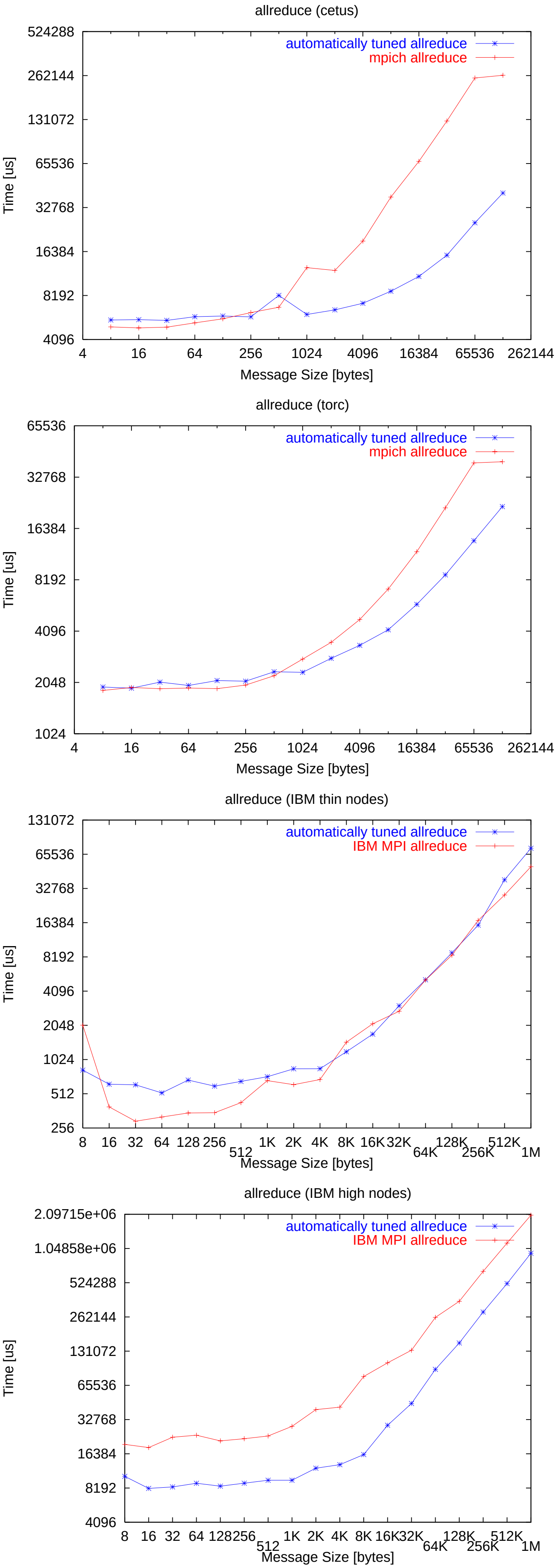


Figure 11: Allgather

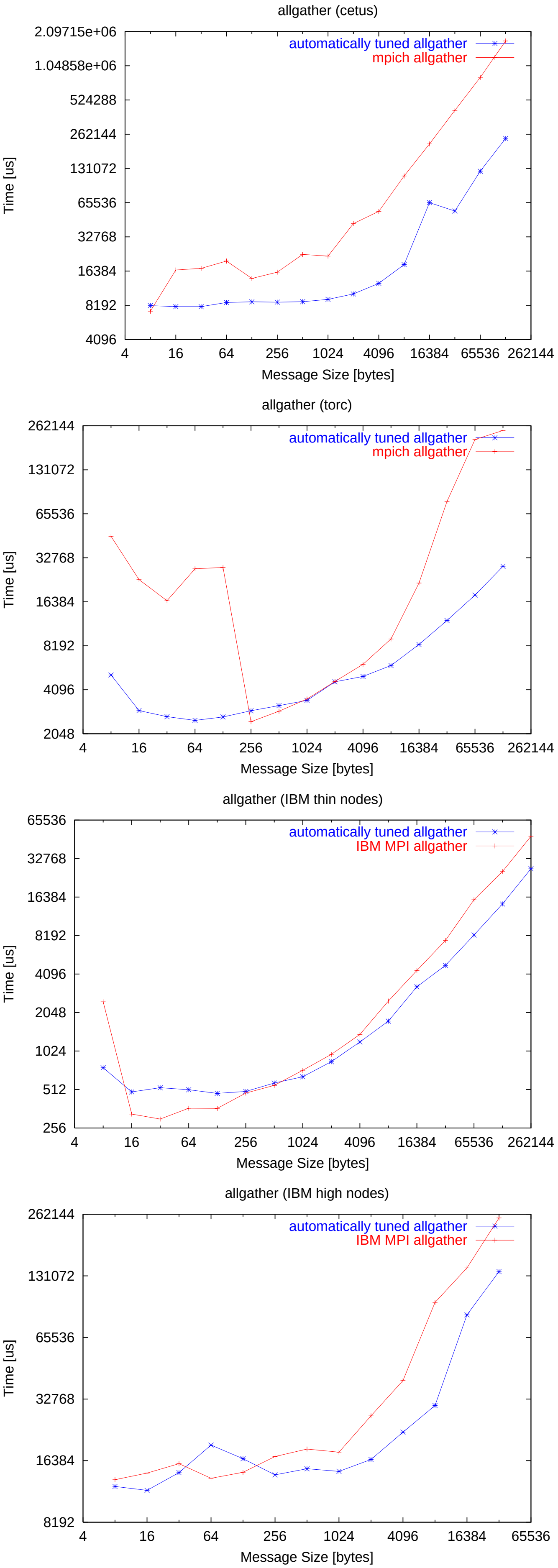


Figure 12: Alltoall

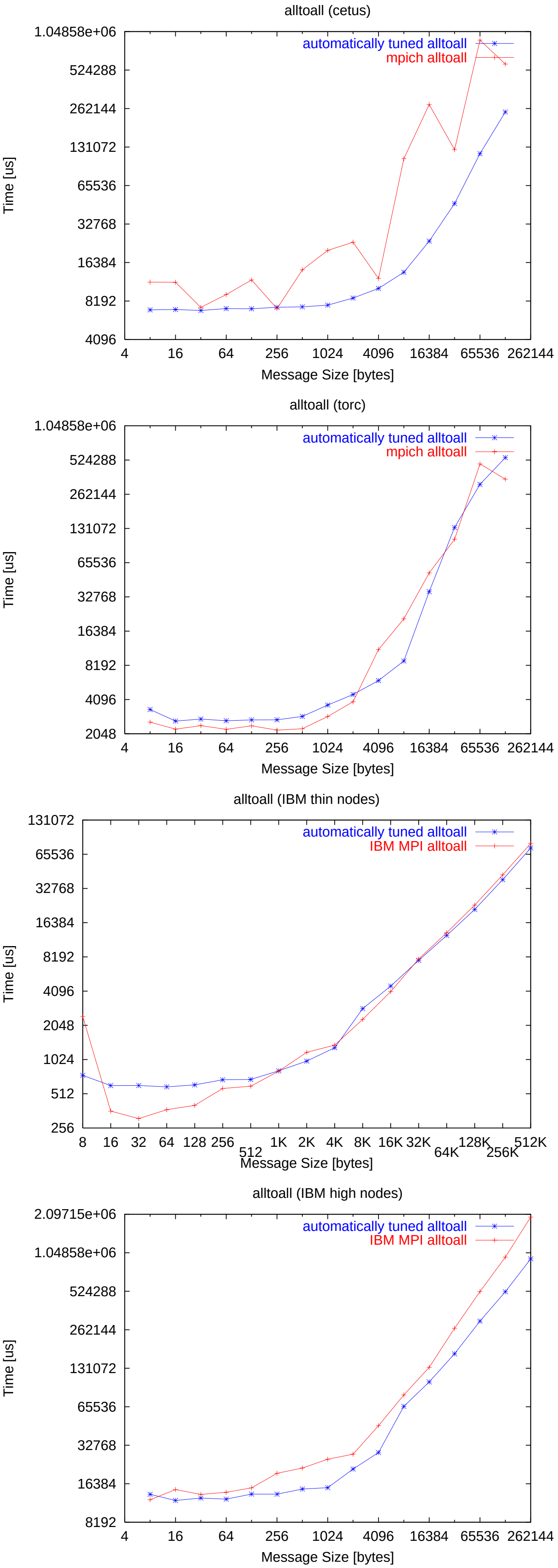


Table 6: Barrier(cetus)

procs	*Distributed binomial*	*Tournament*	*Ring*	*mpich*
2	1225.7	1368.8	1467.3	1090.2
4	2743.8	3119.2	4250.3	2798.5
8	20499.4	22272.4	39757.6	9094.2
16	31136.2	29162.3	48610.6	32712.9

Table 7: Barrier(torc)

procs	*Distributed binomial*	*Tournament*	*Ring*	*mpich*
2	383.7	551.5	501.8	387.0
4	7503.3	1511.6	17200.7	831.8
8	5441.5	9375.0	9457.9	1440.3
16	20126.9	10399.4	17413.3	3400.0

Table 8: Barrier(IBM thin nodes)

procs	*Distributed binomial*	*Tournament*	*IBM MPI*
2	1503.3	2076.8	1836.0
4	3637.9	6138.6	3155.1
8	13671.8	13033.8	14323.5

Table 9: Barrier(IBM high nodes)

procs	*Distributed binomial*	*Tournament*	*IBM MPI*
2	3836.5	129.3	107.0
4	6561.7	214.0	182.8
8	309.6	336.7	290.0

conducted. Even though this only needs to occur once, the time taken for all these experiments can be considerable.

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

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 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.

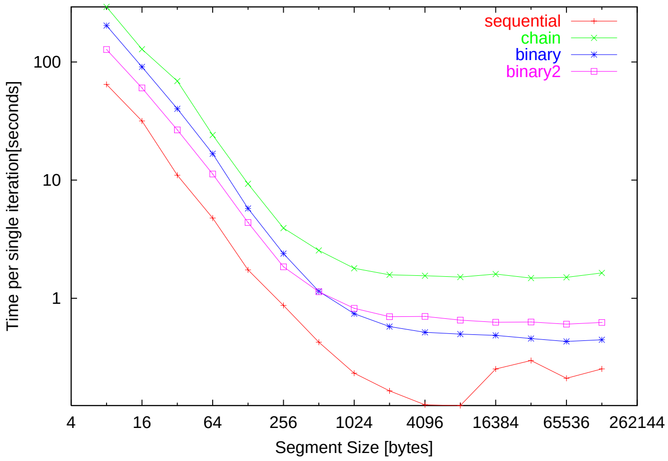
1. Reduced number of grid points with interpolation. For example reducing the message size tests from {8, 16, 32, 64.. 1MB} to {8, 1024, 8192.. 1MB}.
2. Using instrumented application runs to build a table of only those collective operations that are required, i.e. not tuning operations that will never be called, or are called infrequently.
3. Using black box solvers 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} 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}. Figure 13 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 y = exp(-x). 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 13, 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.

Figure 13: Segment size verse time for various communication methods



For this reason, we have developed a number of hill decent 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 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 and thus they can be immediately rejected.

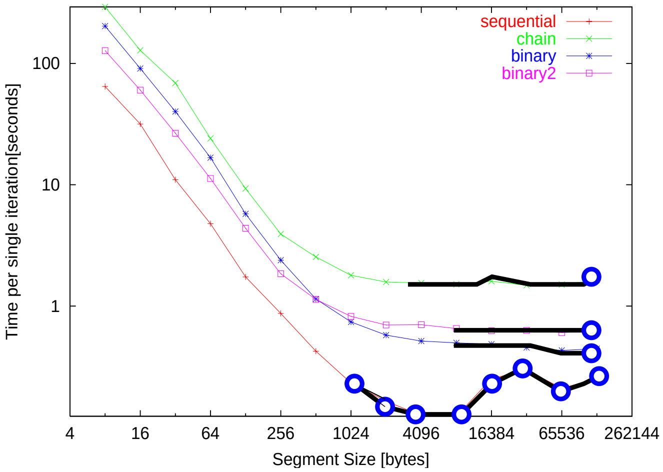
Figure 14 shows the extent of the MGD and SMGD superimposed on the initial scatter results. The MGD extent is marked by thicker lines and the SMGD by individual points.

Table 10 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.

Table 10: Performance of optimizing algorithms)

Method	*Linear Time*	*Linear Iteration*	*MGD Time*	*MGD Iteration*	*SMGD Time*	*SMGD Iteration*	*Speed up*
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

Figure 14:



5. DYNAMIC REORDERING OF TOPOLOGIES

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

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

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

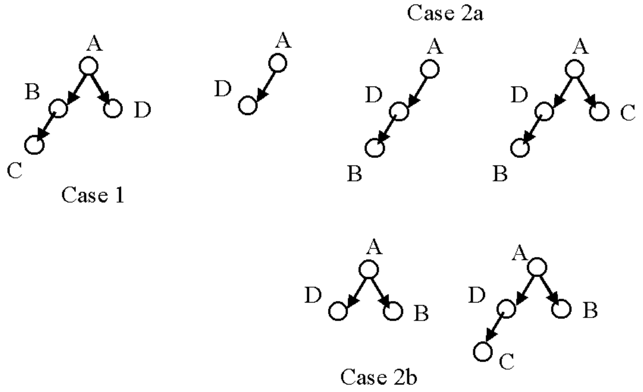
5.1 Dynamic methodology

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

Figure 15 demonstrates this methodology. Case 1 is where all processes within the tree are ready to run immediately and thus performance is optimal. In Case 2, both processes B and C are delayed and initially the root A can only send to D. As B and C become available, they are added to the topology. At this point we have to choose whether to add the nodes depth first as in Case 2a or breadth first as in Case 2b. Currently depth first has given us the best results. Also note that in CASE 1, if process B is not ready to receive, it effects not only its own sub-tree, but depending on the message/segment size, it is possible that it would block any other messages that A might send, such as to Ds sub-tree etc. Faster network protocols might not implement non-blocking sends in a manner that could overcome this limitation without effecting the synchronous static optimal case, and thus blocking send are often used instead.

Currently we are testing the cost of overhead incurred in using this technique for different network infrastructures. We are also exploring the conditions needed for the automatic use of this technique during the course of the computation. Initial results have been promising, especially for large mes-

Figure 15: Reordering a tree topology



sages and network interfaces with very low latency, that rely on the receivers to have already posted receives to allow DMA message transfers. Worst case results have been equivalent to the overhead for n-1 small message send/receives. Best case has been within a few percent of optimal where no re-ordering on the same example has produced multiples of the optimal wall clock times, although this varies with the operation, number of processors, data size and level of initial synchronization.

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

6. CONCLUSION

The optimal algorithm and the optimal buffer size for a given message size depends on a given configuration of the system including the gap values of the networks, memory models, the underlying communication layer etc. The optimal parameters for a system can be best determined by conducting experiments on the system. Our results show that the optimal parameters obtained from the experiments gave better performance than some native MPI implementations which implement a single algorithm irrespective of the system parameters. The randomness of our results for a given system also show that a generalized mathematical model will often not be able to give optimal performance. Also, some modified hill-descent heuristics to reduce the number of experiments were tested. The heuristics gave good performance by reducing the number of experiments by factors of 10-300. We have also shown that during application execution, dynamically altering the mapping between rank and position within a topology can yield additional benefits in terms of performance.

7. FUTURE WORK

More competent algorithms that give good performance on some systems have to be implemented. Good heuristics for conducting less experiments and still being able to obtain optimal performance for a given message size and number of processors have yet to be developed.

8. 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.