

The Analysis and Optimization of Collective Communications on a Beowulf Cluster

Wi Bing Tan and Peter Strazdins,
Department of Computer Science,
Australian National University

Abstract

This paper gives a performance analysis of the All-Gather, All-Reduce and Reduce-Scatter collective communication operations on a Beowulf cluster. This cluster has a contention-free switch-based network with multiple network interface cards per node, permitting overlapping of message transmission under certain circumstances. As well as considering traditional algorithms developed previously for parallel computers with vendor-specific networks, we also examine simpler algorithms made up of repeated sub-operations, such as broadcasts. We find that for the kind of network on the Beowulf cluster, a somewhat different performance modelling of the algorithms is required, and that some simple simulation tools had to be developed in order to fully understand some of the algorithms' performance.

Our results indicate that the LAM MPI implementations for these operations may be significantly improved, and the algorithms with data exchange and potential contention perform well on the cluster. Furthermore, they indicate that algorithms permitting message overlap are slightly favoured, with a new and simple algorithm which modestly out-performs the best traditional algorithms in the case of Reduce-Scatter. With the exception that the degree of overlapping proved difficult to estimate, our performance models fitted closely with the results, and together with the simulation tools, permit a detailed understanding of the cluster's communication pattern performance.

1. Introduction

Collective communication operations, where a group of nodes on a parallel processor must co-operate in a communication, form an important step in many supercomputing applications, including those for dense linear algebra. For example, broadcast and reduction operations are the main communications in LU and QR factorizations in ScaLAPACK [3]; where more sophisticated load balance strategies

are used, some of these operations are replaced by the *All-Gather* and *Reduce-Scatter* operations [5, 1, 10]. These collective operations, and the *All-Reduce* operation, have been considered sufficiently important to have been introduced into the MPI standard ([8]).

Research on this area has established that a variety of algorithms (or communication patterns) exist for such operations, and their performance has been modelled and analysed on mesh and hypercube communication networks on vendor-supplied parallel computers [9, 2, 7]. Here, message contention was often a major issue in the choice of optimal communication patterns.

Recent years have seen the advent of the cluster computers as an alternative model of parallel computer. Typically, their COTS communication networks are relatively slow, and hence the (software) optimization of communication operations becomes more important. Furthermore, these networks tend to be *switch-based*; these are free of contention for messages having disjoint source-destination pairs. Where nodes have multiple Network Interface Cards (NICs), it is also possible that contention can even be avoided between messages sharing a common source or destination. Thus, communication patterns may have significantly different performance characteristics on these networks than on those of proprietary (non-switch) networks.

This paper evaluates the *All-Gather*, *Reduce-Scatter* and *All-Reduce* collective communication operations on such a cluster, a Beowulf x86-based cluster called the *Bunyip*. Its main original contributions are: firstly to measure and analyse these operations' associated communication patterns for such a cluster (in some cases, substantially different performance models are required); secondly to propose a new and simple communication pattern that appears optimal for such clusters; and thirdly to introduce some simulation tools that are useful aids in understanding communication performance.

More recent related work includes a comparison of the ring algorithm-based (see Section 3.3.4) collective communications and vendor-supplied MPI on the Cray T3E and

IBM SP [7], a comparison of all-reduce algorithms on a SCI cluster [6], and an evaluation of automatic tuning of all-reduce and all-gather algorithms on a variety of platforms including a Beowulf-style cluster [4]. However, none of these papers give performance models for these operations, or a comparison of these with experimental results.

This paper is organized as follows. Section 2 describes the Bunyip configuration. Section 3 describes the above collective communication operations and their implementations in terms of various communication patterns, together with the associated performance models appropriate for the Bunyip's network. Results for intra-group communication patterns on the Bunyip given in Section 4, and include comparisons of actual performance with that predicted by the performance models. We extend the above work for intra-group communications in Section 5. A description of the simulator tools is given in Section 6, with conclusions being given in Section 7.

2. The Bunyip Cluster

The Beowulf cluster *Bunyip* used in our experiments is a 96 node dual processor Pentium III running at 550 MHz.

Bunyip consists of 4 groups made up of 24 nodes each. The nodes each has three 100 Megabit NICs, each being connected to an intergroup switch (Hewlett Packard ProCurve 4000M); thus every node is directly connected to every other through one of these switches.

The unique topology of Bunyip allows certain advantages over a classic channel-bonded Beowulf cluster. One advantage is the contention free nature of the network within a group. Each node in a group is connected to its peers by 3 connections.

Another advantage is a degree of overlap where each node is able to communicate with (in particular, receive from) a maximum of 3 other nodes simultaneously. This advantage is evident in algorithms with a series of multiple sends or receives (i.e. *Full Fan-in*, see Section 3.3.7)).

The last advantage this topology provides is the ability to simultaneously send and receive. Algorithms which rely on simultaneous exchange of data (i.e. *Bi-Directional Exchange*, see Section 3.3.1)) should perform well in Bunyip.

3. Algorithm Description and Performance Models

This section describes the communication operations of interest, and the corresponding algorithms (communication patterns) that can be used to implement them. Figure 1 lists notations that are used throughout this section. Note that n corresponds to the (characteristic) message size used in the communication patterns. It is in units of the size of a

double precision number (8 bytes), as reduction operations are typically performed on double precision data.

Variable	Value
α	startup cost
β	communication volume cost
p	number of processors
n	initial number of elements for <i>all-gather</i> and <i>all-reduce</i> , final number of elements for <i>reduce-scatter</i>
$S(n)$	Send time for n elements
$R(n)$	Receive time for n of elements
t_{alg}^{op}	time of operation (op) using algorithm (alg)
ag	All-Gather operation
ar	All-Reduce operation
rs	Reduce-Scatter operation

Figure 1. Notations used in the performance model

3.1. Performance model for Point-to-point Messages

The general timing model used for a single point-to-point messages is:

$$t^{p2p}(n) = \alpha + \beta n \quad (1)$$

When timing message performance on the Bunyip, it was noticed that there is a large difference between the send and receive times. For the accurate modelling of communication patterns involving multiple sends (and to a lesser extent) receives per node, it is sometimes necessary to model the communication times for the sender and receiver separately:

$$S(n) = \alpha_s + \beta_s n, \quad R(n) = \alpha_r + \beta_r n$$

where: $\alpha = \alpha_s + \alpha_r$, $\beta = \beta_s + \beta_r$. On the Bunyip, modified ping-pong tests have revealed the values $\alpha_s = 24\mu\text{s}$, $\alpha_r = 180\mu\text{s}$, $\beta_s = 0.0825\mu\text{s}$ and $\beta_r = 1.063\mu\text{s}$.

3.2. Operations

The operations implemented by the algorithms are those employed by the MPI Standard [8].

3.2.1 All-Gather

An All-Gather starts with each node having n elements of data. The individual data in each node is combined to produce a result of np elements of data. This result of np elements is stored in every node.

3.2.2 All-Reduce

An All-Reduce starts with each node having n elements of data. The data from each node is combined with every other node using a cumulative operation (i.e. maximum, sum, multiplication). The final result of n elements is stored in every node.

3.2.3 Reduce-Scatter

A Reduce-Scatter starts with each node having np elements of data. The data is combined using a cumulative operation (just like All-Reduce). The final data is then segmented with each node receiving n elements of the result.

3.3. Communication Patterns

The selection of the algorithms to analyse were based on three factors: speed, simplicity and the understanding they can yield of the Bunyip's communication network. Diagrams illustrating these algorithms may be found in a companion paper [12].

The algorithms for Sections 3.3.1 to 3.3.4 represent 'classical' collective communication patterns, and are described in [2, 9]. With the exception of the Ring pattern, these tend to have more complex implementations, especially for the case where p is not a power of 2. Most of the performance models for these have appeared in the above references, although in our presentation below we have adapted these for a contention-free communication network, resulting in considerable simplification in some cases. In particular, for the patterns based on exchanges or ring operations, it is assumed that a node can simultaneously send and receive; otherwise the models may be underestimating performance by a factor of up to 2.

The remainder of the patterns are derived from broadcast, (one-)reduce or (one-)gather sub-operations, repeated p times. These have a relatively simple implementation. Whether on a Bunyip-like network communications between the sub-operations can overlap (*sub-operation overlap*), i.e. whether the cost of the repeated operation can be less than p times the cost of that sub-operation, is a question of interest that we address in this paper.

Note that for these patterns, the performance models for All-Reduce are the same as for Reduce-Scatter, i.e. $t_*^{ar}(n, p) = t_*^{rs}(n, p)$, and for the sake of brevity, may be omitted.

3.3.1 Bi-Directional Exchange

Bi-directional Exchange works by simultaneous exchange of data between nodes. In the first stage, the initial data that the node has is sent to its neighbour and the collected data is combined. Then, nodes at distance $d = 4$ exchange

and combine data. As the final stage at distance $d = \frac{p}{2}$ completes, each node has the full combined data.

The following formulae for bi-directional exchange assumes a contention free network capable of supporting simultaneous send and receive:

$$t_{\text{bidir}}^{ag}(n, p) = \alpha \log_2 p + (p - 1)(\beta n) \quad (2)$$

$$t_{\text{bidir}}^{ar}(n, p) = \log_2 p(\alpha + \beta n) \quad (3)$$

As the Bunyip meets the above conditions, and this is an inherently efficient communication pattern, we expect it to yield good performance. For Reduce-Scatter, this pattern is equivalent to Recursive-Halving, so that $t_{\text{bidir}}^{ar}(n, p) = t_{\text{rec}}^{ar}(n, p)$ (see Equation 6).

3.3.2 Recursive-Halving Recursive-Doubling

In the case of All-Reduce, Recursive-Halving Recursive-Doubling works by simultaneous exchange of data between nodes (similar to Bi-Directional Exchange).

In the first stage, the data in the nodes are split in halves and alternate halves are exchanged between nodes (i.e. node 0 exchanges the upper half of its data with the lower half data of node 1). The data received is combined with the node's corresponding data (i.e. node 0 received lower half data from node 1 and combines it with its lower half data). The combined data is further halved; alternate halves are again exchanged between the next node (i.e. node 0 exchanges halves with node 2). This exchanged data is further combined, halved and exchanged again (i.e. node 0 exchanges with node 4). This continues until each node has a unique portion of combined data.

The nodes then proceed with Recursive-Doubling. The unique portion of the combined data is exchanged in an inverse fashion from Recursive-Halving. The received data is combined with the unique portion to form a new combine data. This combine data is sent to the next node. This proceeds until all the nodes have the complete combined data.

In the case of All-Gather, the data is added to form a larger array. The data size of each exchange of data in All-Gather would not change (increasing data width, decreasing data depth). In the case of Reduce-Scatter, Recursive-Halving Recursive-Doubling only performs Recursive-Halving. After Recursive-Halving, each node contains a segment of data which is what Reduce-Scatter does.

Recursive-Halving, Recursive-Doubling may be applied to all three operations:

$$t_{\text{rec}}^{\text{ag}}(n, p) = \log_2 p \left(\alpha + \beta \frac{n}{2} \right) + \alpha \log_2 p + (p-1)(\beta n) \quad (4)$$

$$t_{\text{rec}}^{\text{ar}}(n, p) = 2 \left(\alpha \log_2 p + (p-1) \left(\beta \frac{n}{p} \right) \right) \quad (5)$$

$$t_{\text{rec}}^{\text{rs}}(n, p) = \alpha \log_2 p + (p-1)(\beta n) \quad (6)$$

Due to its small communication volume, Recursive-Halving Recursive-Doubling should perform well in a contention-less network.

3.3.3 Fan-in Fan-out

Fan-in Fan-out is similar to Bi-Directional Exchange in structure but does not exchange data and hence performs better on networks with contention. Fan-in Fan-out first does a combine of the data in an inverted binary-tree like manner. Once the root node has collected the combined data, the data is sent to all the other nodes via a tree broadcast.

Fan-in, Fan-out may be applied to All-Gather and All-Reduce operations, with the following performance models:

$$t_{\text{fan}}^{\text{ag}}(n, p) = \alpha \log_2 p + (p-1)\beta n + \log_2 p (\alpha + \beta np) \quad (7)$$

$$t_{\text{fan}}^{\text{ar}}(n, p) = 2(\log_2 p)(\alpha + \beta n) \quad (8)$$

Fan-in, Fan-out should not perform as well for Bunyip due to its higher communication volume costs.

3.3.4 Ring

For the Ring communication pattern, the nodes start by sending their data to the their successive neighbour node $(\text{rank} + 1) \bmod p$ and receive data from the previous node $(p + \text{rank} - 1) \bmod p$. The received data is combined and sent on to the successive neighbour node. This is repeated for $p - 1$ steps, at which point every node has the complete combined data.

The Ring pattern is contention-free and may be applied to all three operations, with the following performance models:

$$t_{\text{ring}}^{\text{ag}}(n, p) = t_{\text{ring}}^{\text{rs}}(n, p) = t_{\text{ring}}^{\text{ar}}(n, p) = (p-1)(\alpha + \beta n) \quad (9)$$

Ring should perform well for both All-Gather and Reduce-Scatter operations, as its coefficient for β is optimal; however this is not the case for All-Reduce.

3.3.5 Repeated Binary Tree

In the case of All-Gather, Repeated Binary Tree is a series of binary tree broadcasts. The first node broadcasts its data to the other nodes relative to its rank. The second then does the same and so forth.

In the case of Reduce-Scatter, it is the inverse. The first node collects its segment of data using an inverted binary tree one-reduce operation. The second then does the same and so forth.

For the All-Gather, Repeated Binary Tree has some sub-operation overlap. Consider the second broadcast sub-operation, with the root at node 1. For $p \geq 4$, node $\frac{p}{2} + 1$ will be ready to begin the receive from node 1 at time $\log_2 p \cdot S(n) + 2R(n)$, with node 1 being able to initiate the send at time $S(n)$ earlier. Thus we can deduce that the total execution time must be at least $\log_2 p \cdot S(n)(p-1) + 2R(n)$. From time charts generated by the simulator (see Section 6) it can be shown that this degree of overlap can be sustained [12, 11].

However, for the Reduce-Scatter, no such overlap can occur, due to the stringent dependencies inherent in reduce operations. Thus, the performance models for All-Gather and Reduce-Scatter are given as follows:

$$t_{\text{tree}}^{\text{ag}}(n, p) = (p \log_2 p - (p-1))S(n) + \min[\log_2, 2]pR(n) \quad (10)$$

$$t_{\text{tree}}^{\text{rs}}(n, p) = p \log_2 p (\alpha + \beta n) \quad (11)$$

Repeated Binary Tree patterns are of interest because these effectively form the main vertical communications in matrix factorizations and reduction computations [3]. Moderate performance for All-Gather is expected not only on the Bunyip but on any network, as no assumptions of a contention-free network or simultaneous send and receive capability is required for the result of Equation 11.

3.3.6 Pipeline

In the case of All-Gather, Pipeline is a series of pipelined broadcasts. The first node sends its data to the second, its rightward neighbour. This node passes on this message to its rightward neighbour, and so forth until all nodes have received the message from the first node.

As soon as the second node has passed the data from the first, it similarly initiates its own broadcast. This continues until all nodes have initiated their broadcasts; thus, there will be a degree of overlap between the broadcasts.

In the case of Reduce-Scatter, it is similar except each node adds its portion of the destination data to the combined data before passing it on.

This pattern has a start-up sequence of time $\alpha + \beta n$, a broadcast sequence of the same time and a finishing sequence of the same time, yielding the performance model:

$$t_{\text{pipe}}^{ag}(n, p) = t_{\text{pipe}}^{rs}(n, p) = 3(p-1)(\alpha + \beta n) \quad (12)$$

Pipeline is not therefore expected to perform as well as Ring. However, its study here is still of interest, since in computations such as ScaLAPACK matrix factorizations, the major horizontal communications form effectively a repeated pipeline broadcast; in this case, the effective coefficient is 2 rather than 3 (the finishing sequence is effectively removed). Using the *lookahead* technique, in certain circumstances this can be reduced to the (nearly) optimal factor of 1 [10].

3.3.7 Full Fan-in

Full Fan-in is a new pattern, suitable for contention-free networks, which to our knowledge has not been discussed so far in literature. Full Fan-in has p stages, each of which, in the case of All-Reduce, can be thought of as a (one-)gather operation. For each stage, a node will do a series of $p-1$ receives. The order of sending nodes are relative to the receiving node's rank: the first node to send is at $(\text{rank} + 1) \bmod p$ and the next is at $(\text{rank} + 2) \bmod p$, etc.

The performance model for Full Fan-in can be derived with the aid of the Timing charts created using our simulator (Figure 7, Section 6). It can be seen that node 0 completes at time $pS(n) + (p-1)R(n)$, with node i completing at time $iR(n)$ later, $0 \leq i < p$, thus yielding:

$$t_{\text{fi}}^{ag}(n, p) = t_{\text{fi}}^{rs}(n, p) = pS(n) + 2(p-1)R(n) \quad (13)$$

Full Fan-in is simple and should run moderately well on contention-free networks, such as on the Bunyip; it also holds some scope for overlapping multiple receives using the three NICs.

A still simpler variant of this, which we call Full Fan-Out, decouples the order in which the receives occur. That is, all nodes perform all their sends before any start receiving, in a similar pattern to an all-to-all communication. While this may stress the capacity of the network in that up to $p-1$ un-received messages may need to be buffered, it eliminates the dependencies of Full Fan-in, and permits a maximal degree of overlapping between the three NICs. Its performance may be modelled as follows:

$$t_{\text{fo}}^{ag}(n, p) = t_{\text{fo}}^{rs}(n, p) = \frac{p-1}{o}(\alpha + \beta n) \quad (14)$$

where $1 \leq o \leq 3$ is the overlap coefficient due to having three NICs per node (to be determined empirically).

4. Results

This section describes the implementation of the algorithms described above, and their resulting performance. The algorithms were coded in C using LAM-MPI version 6.3.2. Communication between nodes was asynchronous and mainly consists of `MPI_Isend()` and `MPI_Irecv()`. The reduce operations use addition to combine elements; this occurs sufficiently fast relative to message transfer on the Bunyip that its effect on performance may be neglected. Bunyip nodes within a single group were used in all measurements.

Figures 2, 3 and 4 show the results obtained from timing the algorithms. The element size is that of a double precision word (8 bytes), and $p = 8$ processors have been used. Note that to aid readability, the keys list the communication patterns in order of increasing performance at $n = 1000$.

In Figures 2 and 3, Fan-in Fan-out has the same performance as the generic MPI implementation. From this, we conclude that Fan-in Fan-out is used by MPI. Similarly in Figure 2, Ring has the essentially the same performance as Bi-Directional Exchange, and a separate line is not shown.

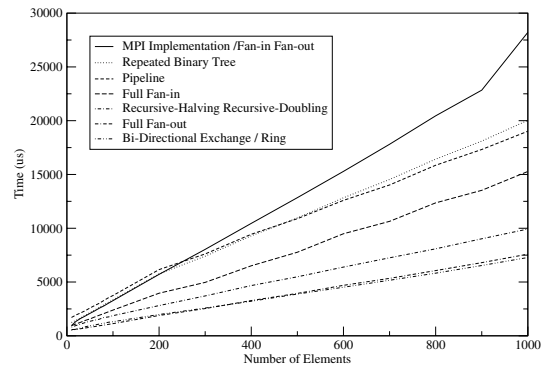


Figure 2. Performance of the All-Gather operation

Figure 5 indicates All-Gather performance over an increasing number of processors. The expected scalability is demonstrated similarly for the All-Reduce and Reduce-Scatter operations [11].

4.1. Discussion of the Results

The results bear out the assumption of good scalability as predicted in the timing models in section 3.3. In particular, the performance of the exchange and ring-based patterns verifies that nodes can simultaneously send and receive a message without significant performance degradation. The results also demonstrate that for $n \geq 200$, communication

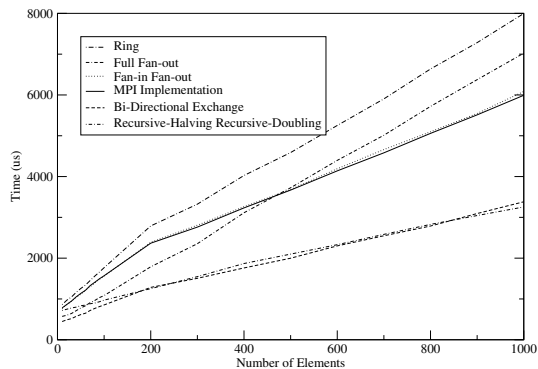


Figure 3. Performance of the All-Reduce operation

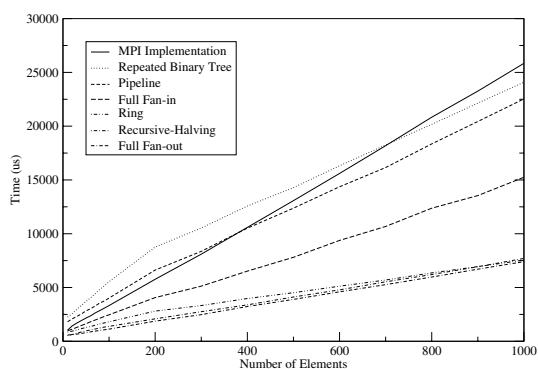


Figure 4. Performance of the Reduce-Scatter operation

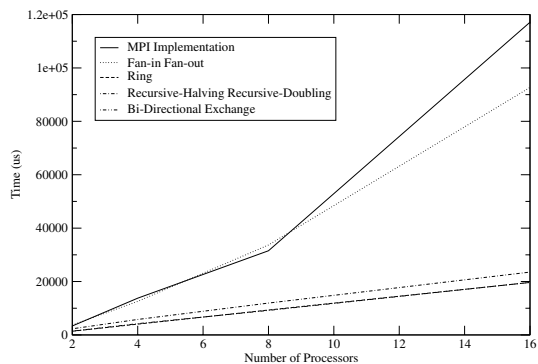


Figure 5. Performance for $n = 1000$ over increasing number of processors

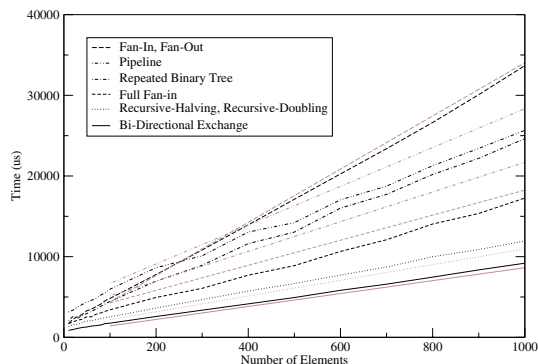


Figure 6. Fitting for All-Gather Results

volume costs become dominant, and the algorithms optimal in this respect yield the best performance.

Thus, Recursive-Doubling Recursive-Halving, Bi-Directional Exchange and Full Fan-in perform well for all operations. Ring performs well for All-Gather and Reduce-Scatter but does not do well for All-Reduce. Other observations from the results also seems to indicate that MPI uses Fan-in Fan-out as its algorithm.

Measurements were also performed for larger messages ($1000 \leq n \leq 10000$) [11]. The trends were very similar, except that there was some degradation of the Full Fan-In and Fan-out patterns, which involve simultaneous multiple receives (in particular a ‘jump’ at $n = 8000$ for both was observed). The Ring pattern’s performance also degraded, as did MPI, for All-Gather.

4.2. Comparison with Performance Models

Figure 6 shows the fit for All-Gather at $p = 8$. In general, the algorithms without overlap fit well though those with overlap do not fit as well. This is especially evident in Full Fan-in and the anomaly is caused by simultaneous receives which Full Fan-in and Pipeline cater for. This anomaly will be further described in Section 6.1 where the diagramming tool is used to understand overlap better.

With a value of $\sigma = 1$ for All-Gather, and $\sigma = 1.2$ for Reduce-Scatter, the performance models for the Full Fan-out pattern of Section 3.3.7 can be made to the results precisely, and so are omitted from Figure 6.

5. Inter-Group Communication Patterns

In this section, we extend our study to communication patterns involving multiple groups on the Bunyip. The most obvious topological difference is in the potential for overlapping of multiple receives of a node: if the messages are from outside the group, they must be from separate groups

for overlap to be possible. However, as seen in Section 4, this factor has at most small impact on the results and performance modelling.

The Bunyip nodes may be thought of as forming 4×24 processor grid, with 4 groups of 24 nodes. As mentioned in more detail in [11], the following approaches were tried: a direct implementation of the unmodified intra-group algorithm (with the MPI rank corresponding to a row-major or a column-major ordering of the 4×24 grid), and a hierarchical one (communicate across rows first, then along columns; and visa-versa). The Ring and Bi-Directional Exchange patterns, being the best all-round performance for intra-group communications, were used as a basis.

Results for All-Gather on a 4×16 sub-grid indicate that all methods were near identical for $n \leq 1000$, but all methods except the hierarchical approach (communicating across columns first) suffered a reduction of bandwidth by up to 20% for $1000 \leq n < 10000$. This was because only this approach avoided larger inter-group messages.

6. Simulator

The simulator is a collection of programs written to help understand the interaction between nodes during communication. It worked mainly as a debugging tool and secondly as a resolution tool between the predicted and actual results.

The software is made up of two parts, the simulator program proper and the diagramming tool. The simulator uses MPI to resolve the sequence of send and receive events and generates the diagrams based on theoretical data (results from ping-pong). The diagramming tool also uses MPI to resolve the sequence of send and receive events but also uses MPI to give the start time in order to generate the diagrams.

An additional library was added to MPI which implements a customised version of `MPI_Isend()` and `MPI_Irecv()`. This customised version adds information tags (including node ids, message size and timestamps) to the output before doing the send / receive. This output is used by the simulator and the diagramming tool to generate the results, with the aid of a parser and sorter.

The simulator program takes the information stored in the tokens and uses α_s , α_r , β_s and β_r values given as input to generate a time chart. The simulator assumes the time at the end of the previous command is that of the start of the next command.

The diagramming tool is similar to the simulator except the start time of the command is calculated using the tags generated from MPI. The diagramming tool displays the actual start of a send/receive call, with the end of send or receive being predicted by pre-defined α and β values (it thus does not take into account idle time).

The software is generic and portable; it can be downloaded from [12].

6.1. Example

Figure 7 shows an example of the simulator output for the Full Fan-in pattern, which was used to derive the performance model of Equation 13.

The measured time chart of the operation, measured by the Diagramming Tool, indicates a squeezing effect where some overlap on multiple receives occurred for the first few nodes (see the companion document from [12]). Due to not taking account of this effect, the performance model of Equation 13 underestimates performance by 10–20

7. Conclusions

Our results show that for clusters with communication networks such as the Bunyip, 'generic' libraries such as LAM MPI are unlikely to provide the optimum algorithms for collective communications, and tuning and experimentation may be required. In this case, we were able to find algorithms whose performance exceeded that of LAM MPI by a factor of between 2 and 3.

For such networks, the Binary-exchange and Recursive Halving algorithms performed well for small messages, due to their optimal α coefficients, and were optimal or close to optimal for larger messages, due to the contention-free nature of the network. For larger messages, the simple Ring algorithm proved as efficient for All-Gather.

We also have investigated communication patterns involving repeated, simple sub-communications. In the case of the Repeated Binary Tree pattern, overlap occurred between consecutive broadcasts, resulting in the overall cost being significantly less than the expected $p \log_2 p$ times that of a point-to-point message. The Pipeline pattern performed better, and would especially do so when embedded into applications in which some of the communications are naturally pipelined, such as the row communications in matrix factorizations.

We have also found that for patterns where messages may be overlapped, e.g. the Repeated Binary Tree and Full Fan-in patterns, accurate performance modelling requires send and receive times to be considered separately. Furthermore, the relatively simple simulation and diagramming tools that we have developed proved helpful in understanding how and to what extent overlap occurred.

A new pattern, which is the simplest of all, called Full Fan-out, proved to be as fast or modestly faster than the best traditional algorithms, for the All-Gather and Reduce-Gather operations. This pattern permitted some overlap between receives sent to the same destination node due to the nodes having multiple NICs. However, it is not clear at this

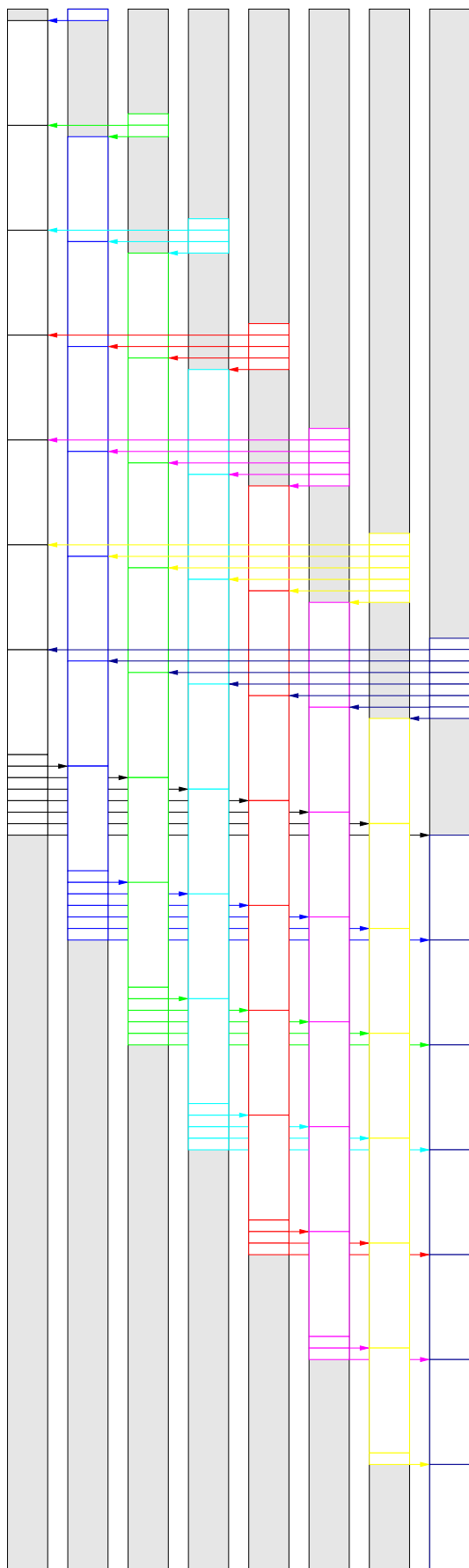


Figure 7. Time Chart for Full Fan-in (Simulator)

stage why the degree of overlap was as small as was observed on the Bunyip; possibly, it might be greater on other clusters with contention-free networks and multiple NICs.

For intra-group communications, similar results applied as for intra-group communications, except that hierarchical methods, minimizing large messages between groups, had a slight performance advantage.

Future work includes the use of TCP/IP multicasts, which appears to be supported by Bunyip's switches, in order to improve All-Gather performance; a preliminary implementation indicates that the Bunyip's switches do indeed support this, but did not always deliver broadcasts reliably [11].

References

- [1] G. Baker, J. Gummels, G. Morrow, B. Riviere, and R. van de Geijn. PLAPACK: High Performance through High Level Abstraction. In *Proceedings of ICPP98: The 27th International Conference on Parallel Processing*, 1998.
- [2] M. Barnett, R. Littlefield, D. Payne, and R. van de Geijn. On the Efficiency of Global Combine Algorithms for 2-D Meshes with Wormhole Routing. *JDPDP*, 1995.
- [3] L. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, J. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. Whaley. *SciLAPACK User's Guide*. SIAM Press, Philadelphia, 1997.
- [4] S. S. V. G. E. Fagg and J. Dongarra. Automatically Tuned Collective Communications. In *Proceedings of Supercomputing 2000*, Oct. 2000.
- [5] B. Hendrickson and D. Womble. The Torus-Wrap Mapping for Dense Matrix Calculations on Massively Parallel Computers. *SIAM J. Sci. Stat. Comput.*, 15(5):1201–1226, 1994.
- [6] L. P. Huse. Collective Communication on Dedicated Clusters of Workstations. In *Proceedings of 6th PVM / MPI European Users Meeting (EuroPVM/MPI'99)*, Sept. 1999.
- [7] G. R. Luecke, B. Raffin, and J. J. Coyle. The Performance of the MPI Collective Communication Routines for large messages on the Cray T3E-600, the Cray Origin 2000, and the IBM SP.
- [8] Message Passing Interface Forum. *MPI: A Message Passing Interface Standard*, jun 1995.
- [9] P. Mitra, D. Payne, L. Shuler, R. van de Geijn, and J. Watts. Fast Collective Communication Libraries, Please. In *Proceedings of the Intel Supercomputing Users' Group*, 1995.
- [10] P. Strazdins. A Comparison of Lookahead and Algorithmic Blocking Techniques for Parallel Matrix Factorization. *International Journal of Parallel and Distributed Systems and Networks*, 4(1):26–35, Apr. 2001.
- [11] W. B. Tan. Analysis and Optimization of Communication Patterns on a Beowulf. Honours Thesis, Department of Computer Science, Australian National University, July 2002.
- [12] W. B. Tan and P. Strazdins. Collective Communications on a Beowulf project. <http://cs.anu.edu.au/~Peter.Strazdins/projects/ClusterComm>, Aug. 2002.