Collective Communication on Architectures that Support Simultaneous Communication over Multiple Links

Ernie Chan Robert van de Geijn
Department of Computer Sciences
The University of Texas at Austin
{echan, rvdg}@cs.utexas.edu

William Gropp Rajeev Thakur
Mathematics and Computer Science Division
Argonne National Laboratory
{gropp, thakur}@mcs.anl.gov

Abstract

Traditional collective communication algorithms are designed with the assumption that a node can communicate with only one other node at a time. On new parallel architectures such as the IBM Blue Gene/L, a node can communicate with multiple nodes simultaneously. We have redesigned and reimplemented many of the MPI collective communication algorithms to take advantage of this ability to send simultaneously, including broadcast, reduce(-to-one), scatter, gather, allgather, reduce-scatter, and allreduce. We show that these new algorithms have lower expected costs than the previously known lower bounds based on old models of parallel computation. Results are included comparing their performance to the default implementations in IBM's MPI.

Categories and Subject Descriptors D.m [*Software*]: Miscellaneous

General Terms Algorithms, Performance

1. Introduction

Extensive research over the past decade has been reported on collective communication and implementations of algorithms for distributing data between processors [6, 7, 9, 13, 14, 15, 16, 20, 25, 27, 28, 29, 30, 33]. It has been shown that effective communication algorithms can be implemented with simple yet powerful techniques [2, 3, 4, 5, 10, 12, 17, 18, 21, 22, 31, 32]. Those techniques inherently assumed that a single node can at most send and receive with one other node at a time. That assumption gave rise to many algorithms, including bidirectional exchange algorithms such as recursive-doubling and halving. Many of those algorihtms are optimal for either startup costs, per data transmission time, or both, based on old models of parallel computation given the constraint on communication.

Given the advent of new parallel architectures, new models of parallel computation can be developed that dramatically decrease the perceived lower bounds of collective communication. We have revisited, redesigned, and reimplemented to take advantage of the new feature where a single node can communicate with multiple nodes simultaneously in order to achieve the new lower bounds of communication. In practice, our implementations show a performance increase of up to a factor of eight from IBM's MPI default collective implementations.

Frequently, collective communication involving all nodes is required. Examples include simple collective communications, such as broadcast, and more complex ones like the various reduction operations. These operations are generally implemented by individual calls to the send and receive routines.

We use broadcast (Bcast) as a motivating example, which can be described as follows: initially a single node, the root, owns a vector of data, x, of length n. Upon completion, all nodes own a copy of x. All operations are illustrated in Fig. 1.

The rest of the paper is organized as follows. In Section 2 we describe our new model of parallel computation on N-dimensional tori where a node can communicate over multiple links. Given that new model, lower bounds for each collective operation are given in Section 3. We present the generalization of several well-known algorithms adapted for the new model in order to achieve the lower bounds in Section 4. Descriptions of the IBM Blue Gene/L and its compability to communicate to multiple nodes simultaneously are given in Section 5.1. We provide performance results of the implementations of our new algorithms in Section 5.2.

2. Model of Parallel Computation

To analyze the performance of the presented algorithms, it is useful to assume a simple model of parallel computation. The following assumptions are made in this report:

Target architectures: The target architectures are distributed-memory parallel architectures.

Indexing: The parallel architecture contains p computational nodes (nodes hereafter). The nodes are indexed from 0 to $p-1$. Each node has one computational processor.

Logically fully connected: Any node can send directly to any other node where a communication network provides automatic routing.

Topology: The underlying topology is an N-dimensional torus. Each node is directly connected to each of its $2N$ nearest neighbors where two are on opposing sides of a dimensional axis.

Communicating between nodes: At any given time, a single node can send *or* receive messages from $2N$ other nodes. A single node can do so *simultaneously* only if each of the messages are sent or received on each of its $2N$ different links.

Cost of communication: The cost of sending a message between two nodes will be modeled by $\alpha+n\beta$, *in the absence of network conflicts.* Here α and β respectively represent the message

PPoPP'06 March 29–31, 2006, New York, New York, USA.

Operation	Before	After
Broadcast	Node 0: x	Node 0: x; Node 1: x; Node 2: x; Node 3: x
Reduce(-to-one)	Node 0: $x^{(0)}$; Node 1: $x^{(1)}$; Node 2: $x^{(2)}$; Node 3: $x^{(3)}$	Node 0: $\sum_j x^{(j)}$
Scatter	Node 0: x_0, x_1, x_2, x_3	Node 0: x_0; Node 1: x_1; Node 2: x_2; Node 3: x_3
Gather	Node 0: x_0; Node 1: x_1; Node 2: x_2; Node 3: x_3	Node 0: x_0, x_1, x_2, x_3
Allgather	Node 0: x_0; Node 1: x_1; Node 2: x_2; Node 3: x_3	Node 0: x_0, x_1, x_2, x_3; Node 1: x_0, x_1, x_2, x_3; Node 2: x_0, x_1, x_2, x_3; Node 3: x_0, x_1, x_2, x_3
Reduce-scatter	Node 0: $x_0^{(0)}$, $x_1^{(0)}$, $x_2^{(0)}$, $x_3^{(0)}$; Node 1: $x_0^{(1)}$, $x_1^{(1)}$, $x_2^{(1)}$, $x_3^{(1)}$; Node 2: $x_0^{(2)}$, $x_1^{(2)}$, $x_2^{(2)}$, $x_3^{(2)}$; Node 3: $x_0^{(3)}$, $x_1^{(3)}$, $x_2^{(3)}$, $x_3^{(3)}$	Node 0: $\sum_j x_0^{(j)}$; Node 1: $\sum_j x_1^{(j)}$; Node 2: $\sum_j x_2^{(j)}$; Node 3: $\sum_j x_3^{(j)}$
Allreduce	Node 0: $x^{(0)}$; Node 1: $x^{(1)}$; Node 2: $x^{(2)}$; Node 3: $x^{(3)}$	Node 0: $\sum_j x^{(j)}$; Node 1: $\sum_j x^{(j)}$; Node 2: $\sum_j x^{(j)}$; Node 3: $\sum_j x^{(j)}$

Figure 1. Collective communications considered in this paper.

startup time and per data transmission time, and n is the total number of bytes communicated.

Network conflicts: The path between two communicating nodes, determined by the topology and the routing algorithm, is completely occupied. Therefore, if some link in the communication path is occupied by two or more nodes, a network conflict occurs. This extra cost is modeled with $\alpha + kn\beta$ where k is the maximum over all links of the number of conflicts on the links.

Cost of computation: The cost required to perform an arithmetic operation is denoted by γ.

3. Lower Bounds

Before discussing collective communication algorithms, it is useful to present lower bounds on the time required to perform an operation in order to to better gauge the algorithms. In this section, we give informal arguments to derive these lower bounds. We will distinguish three components of communication cost: terms due to latency, bandwidth, and computation. Lower bounds are given in Table 1.

Latency: The lower bound on latency is derived by the simple observation that for all collective communications at least one node has data that must somehow arrive at all other nodes. In our model, at each step we can send to $2N$ nodes.

Computation: Only the reduction operations require computation. The computation involved would require $(p-1)n$ operations if executed on a single node or time $(p-1)n\gamma$. Distributing this computation equally among the nodes reduces the time to $\frac{p-1}{p}n\gamma$ under ideal circumstances. Hence the lower bound.

Bandwidth: For broadcast and reduce(-to-one), the root node must either send or receive n items. The cost of this operation is bounded below by $\frac{n\beta}{2N}$ because each node can send or receive to $2N$ other nodes *simultaneously*. For the gather and scatter, the root node must either receive or send $\frac{p-1}{p}n$ items, with a cost of at least $\frac{p-1}{p}\frac{n\beta}{2N}$. The same is the case for all nodes during

Communication	Latency	Bandwidth	Computation
Broadcast	$\lceil log_{2N+1}(p) \rceil \alpha$	$\frac{n\beta}{2N}$	–
Reduce(-to-one)	$\lceil log_{2N+1}(p) \rceil \alpha$	$\frac{n\beta}{2N}$	$\frac{p-1}{p} n\gamma$
Scatter	$\lceil log_{2N+1}(p) \rceil \alpha$	$\frac{p-1}{p} \frac{n\beta}{2N}$	–
Gather	$\lceil log_{2N+1}(p) \rceil \alpha$	$\frac{p-1}{p} \frac{n\beta}{2N}$	–
Allgather	$\lceil log_{2N+1}(p) \rceil \alpha$	$\frac{p-1}{p} \frac{n\beta}{2N}$	–
Reduce-scatter	$\lceil log_{2N+1}(p) \rceil \alpha$	$\frac{p-1}{p} \frac{n\beta}{2N}$	$\frac{p-1}{p} n\gamma$
Allreduce	$\lceil log_{2N+1}(p) \rceil \alpha$	$2\frac{p-1}{p} \frac{n\beta}{2N}$	$\frac{p-1}{p} n\gamma$

Table 1. Lower bounds for the different components of communication cost.

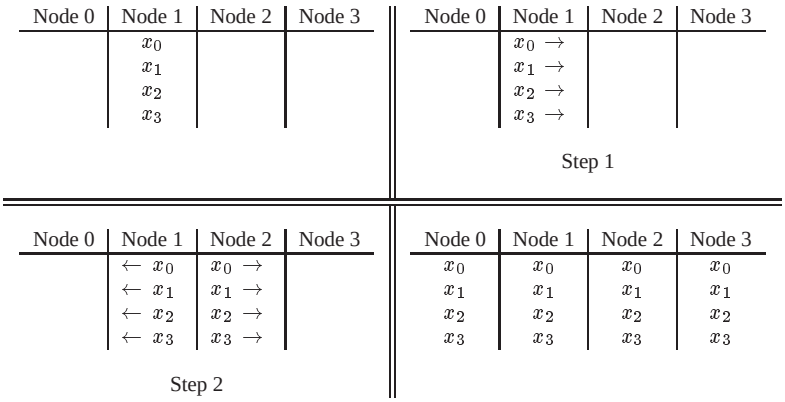

Figure 2. Minimum-spanning tree algorithm for broadcast.

allgather and reduce-scatter. Allreduce is somewhat more complicated. *If the lower bound on computation is to be achieved,* one can argue that $\frac{p-1}{p}n$ items must leave each node, and $\frac{p-1}{p}n$ items must be received by each node after the computation is completed for a total cost of at least $2\frac{p-1}{p}\frac{n\beta}{2N}$.

4. Generalization of Communication Algorithms

Depending on the amount of data involved in a collective communication, the strategy for reducing the cost of the operation differs. When the amount of data is small, the cost of initiating messages, α, tends to dominate, and algorithms should strive to reduce this cost. In other words, the lower bound on the latency in Table 1 becomes the limiting factor. When the amount of data is large, the cost per item sent and/or computed, β and/or γ, becomes the limiting factors. In this case the lower bound due to bandwidth and/or computation in Table 1 is the limiting factor.

There are several classes of algorithms for collective communication, but we concentrate on these two cases: "short-vector algorithms" and "long-vector algorithms." The minimum-spanning tree and bucket algorithms represent the short- and long-vector algorithms, respectively. These algorithms were developed specifically with the assumption that a node can only send and receive from one other node at a time. We present generalizations of those algorithms on N-dimensional tori where each node can send to $2N$ other nodes simultaneously.

4.1 Minimum-Spanning Tree Algorithm

The minimum-spanning tree (MST) algorithm is used for the short-vector algorithms because it organizes communication along the edges of a minimum-spanning tree that cover the nodes involved in the communication. The algorithm requires $O\left(\log(p)\right)$ stages where each contributes the cost of one latency, α, to the total cost of the algorithm. We present the generalization of the MST algorithm that is bounded by $\Theta\left(log_{2N+1}(p)\right)$ on N-dimensional tori.

4.1.1 Original MST Algorithm

On an arbitrary number of nodes, the MST algorithm can be described as follows. Viewing the nodes as a linear array, partition this network into (roughly) two disjoint subnetworks. Send the data from the root to some node, the destination, in the part of the network of which the root is not a member. Recursively continue in the two separate subnetworks, with the original root and the destination as roots for their respective subnetworks. This is illustrated in Fig. 2 for broadcast. The cost of this algorithm is

$$T_{\mathrm{MSTBcast}}(p,n) = \lceil \log_2(p) \rceil (\alpha + n\beta)$$

The MST algorithm operates as a directed graph forming a tree with one incoming and one outgoing edge at each vertex. The height of this tree is $\lceil \log_2(p) \rceil$, hence the number of stages.

4.1.2 MST on N-Dimensional Meshes

The original MST algorithm assumed that the system of nodes was a linear array. We can generalize the MST algorithm on N-dimensional meshes. We assume that the topology is a mesh as opposed to a torus for this algorithm since no messages need to be sent around a ring of any particular dimensional axis. We also assume that we can send or receive N messages simultaneously as long as each message is sent along different dimensional axes. In the next section we further generalize this algorithm to send to the two opposing nodes on each dimensional axis.

Our goal is to partition p nodes into $N+1$ separate partitions in order to form a tree with a height of $\lceil \log_{N+1}(p) \rceil$. One simple

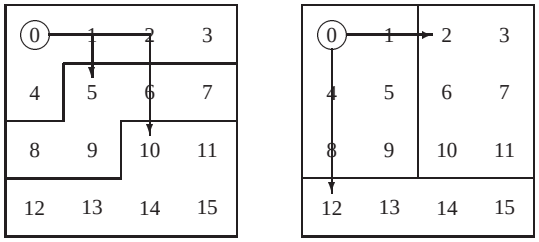

Figure 3. Left: Simple partitioning of a 4 × 4 mesh into three partitions where node 0 is the root sending to two other nodes. Right: Partitioning the mesh using our generalized algorithm.

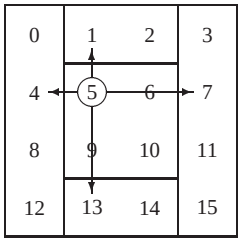

Figure 4. Partitioning a 4 × 4 torus into five partitions where node 5 is the root sending to four other nodes simultaneously.

approach to partition the nodes is to divide p by $N+1$. Even though this approach is relatively simple to implement, data messages might not be sent simultaneously when using this algorithm. We are given the constraint that sending simultaneously can occur only by saturating the outgoing $2N$ links from a node. We illustrate the partitioning of sixteen nodes in a 4 × 4 mesh in the left of Fig. 3. With the first node as the root, given an automatic and usually nondeterministic message routing in this disjointed partitioning, sending simultaneously may not be facilitated. In our example both data messages are routed through node 1, thereby leading to a network conflict.

Another naive approach to partitioning is to recursively subdivide the mesh in half along each dimension. This approach creates an unbalanced N-ary tree with the maximum height of $\lceil \log_2(p) \rceil$ because it is bounded by the first subdivision of the mesh in half. This approach does have the advantage of creating separate cubic partitions that can facilitate sending simultaneously, yet the cost of this algorithm is no better than the original MST algorithm.

To get $N+1$ partitions of roughly equal size, we simply divide the mesh by partitioning off nodes from each dimension by a decrementing counter starting from $N+1$. This partitioning is illustrated in the right of Fig. 3. In our example, we are given a two-dimensional mesh, so divide the first dimension by three to partition off one row from the mesh. In the remaining three rows, we decrement our counter and divide the second dimension by two to partition off two columns, which gives three partitions of sizes 3 × 2, 3 × 2, and 1 × 4.

D is an ordered N-tuple representing the number of nodes in each dimension of the mesh, and D^i represents the partition along the ith dimension where the root does not reside. So the partitioning is described by

for $(i = 1 : N)$
  for $(j = 1 : N)$
    if $(j > i)$
      $D^i_j = D_j$
    else if $(j < i)$
      $D^i_j = D_j - D^j_j$
    else
      $D^i_j = D_j/(N + 2 - i)$

where

$$\prod_{i=1}^{N} D_i = p \text{ and } 0 < i \leq N, D_i > 1 \text{ and } |D| = N$$

This partitioning does not give the entire algorithm, but all indexing of nodes is derived from each partitioning of D. A full description of the algorithm would nearly need the actual code because of the many complicated cases arising from the integer arithmetic.

The MST has a maximum and average height of $\lceil \log_{N+1}(p) \rceil$. The number of nodes in each partition is not as even as simply dividing by $N+1$, so the average height is a rough approximation, but sending simultaneously is almost assured. Reducing to a one-dimensional mesh, this algorithm is the original MST algorithm.

In order to send to any many nodes simultaneously, each partition needs to be as cubic as possible as opposed to elongated strips that eventually get reduced to one dimension. We can simply partition the mesh by the order of the dimensions with the most nodes first.

4.1.3 MST Sending Twice on Each Dimensional Axis

We now deal with sending messages in the two opposing directions on each dimensional line. If dealing with a linear array once again, the MST algorithm can be performed by dividing the nodes into three partitions and then sending to the two opposing partitions. This may require sending around the ring, so our algorithm now makes use of the torus. The algorithm is adjusted by

for $(i = 1 : 2N)$
  for $(j = 1 : N)$
    if $(j > (i-1) \bmod N + 1)$
      $D^i_j = D_j$
    else if $(j < (i-1) \bmod N + 1)$
      $D^i_j = D_j - D^j_j - D^{j+N}_j$
    else
      $D^i_j = D_j/(2N + 2 - i)$

where D^{i+N} represents the partition along the ith dimension on the opposing side of the root from the D^i partition. The MST algorithm completes in $\lceil \log_{2N+1}(p) \rceil$ stages, which attains the lower bound for latency. We illustrate this partitioning in Fig. 4.

If strictly dealing with a mesh, we can simply adjust the algorithm by decrementing the counter from $2N+1$ within the assignment when $j = (i-1) \bmod N + 1$ to start the subdivision by however many dimensions have the root in a noncentral position. Those dimensions where the root is not in a central position of the dimensional line can simply be subdivided in half as opposed to three separate partitions along that axis.

4.1.4 Summary

Given any N-dimensional tori, we can perform the MST algorithm in $\lceil \log_{2N+1}(p) \rceil$ stages instead of just $\lceil \log_2(p) \rceil$, which is an improvement of $\log_2(2N+1)$. As with the original algorithm, the generalized algorithm also does not incur network conflicts. Reduce(-to-one) is implemented in much the same way in reverse where the data is sent to the root.

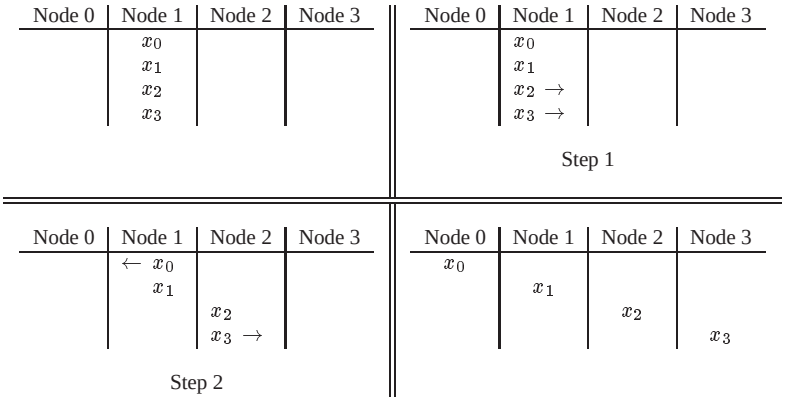

Figure 5. Minimum-spanning tree algorithm for scatter.

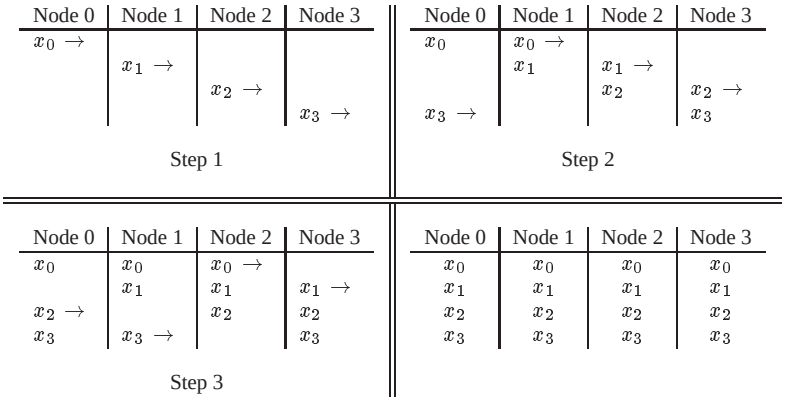

Figure 6. Bucket algorithm for allgather.

4.1.5 Generalized MST Scatter

A scatter can be implemented much like a broadcast using an MST algorithm, except that at each step of the recursion only the data that ultimately must reside in the subnetwork, where the destination is, needs to be sent from the root to the destination. The one-dimensional MST scatter algorithm is illustrated in Fig. 5. Under the assumption that $\log_{2N+1}(p)$ is an integer and all subvectors are of equal length, the cost of this approach is given by

$$T_{\mathrm{MSTScatter}}(p,n,D) = \sum_{k=1}^{\log_{2N+1}(p)} \left(\alpha + (2N+1)^{-k} n\beta\right) =$$

$$\log_{2N+1}(p)\alpha + \frac{p-1}{p}\frac{n\beta}{2N}$$

which attains the lower bound for both latency and bandwidth. Gather can also be implemented much like scatter in reverse. The MST scatter and gather algorithms are used as both the long- and short-vector algorithms since the MST algorithm are optimal for both components of the cost of communication in these operations.

4.2 Bucket Algorithm

When implementing long-vector algorithms, the goal is to reduce the β and γ terms of the cost. We show that a simple building-block approach to the development of algorithms again yields algorithms that are, in theory, near-optimal but for the β and γ terms. The algorithm we present is often referred to as bucket algorithm since it collects data at each node like a bucket during each step of an operation. We generalize this algorithm where N buckets collect data simultaneously.

4.2.1 Original Bucket Algorithm

The bucket algorithm is used to implement either the allgather or reduce-scatter. A simple approach to the implemention of the allgather operation views the nodes as a ring. At each step, all nodes send data to the node to their right. In this fashion, the subvectors that start on the individual nodes are eventually distributed to all nodes. The process is illustrated in Fig. 6. Notice that, if each node starts with an equal subvector of data, the cost of this approach is given by

$$T_{BucketAllgather}(p,n) = (p-1)(\alpha + \frac{n}{p}\beta) =$$

$$(p-1)\alpha + \frac{p-1}{p}n\beta$$

4.2.2 Bucket on N-Dimensional Tori

We generalize the bucket algorithm to N dimensions by gathering data on each of the N different dimensional lines simultaneously. The difficulty lies when trying to gather data messages that do not lie on any dimensional axis as the receiving node. If data on a node was sent to its N different neighbors at each step, then inherent conflicts will occur with data arriving at a node from multiple locations. We eliminate this problem by arbitrarily ordering where the data is sent and received. We illustrate this inherent conflict in

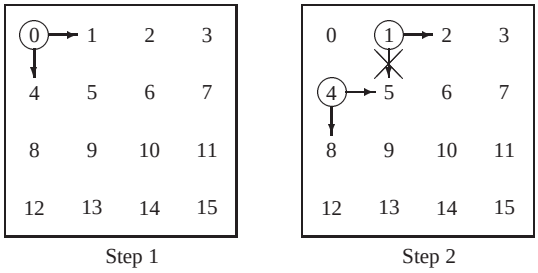

Figure 7. Sending data message x_0 through two stages of the generalized bucket algorithm in a 4 $\times$ 4 torus where a conflict occurs when nodes 1 and 4 try to send the same data to node 5 at the second step.

Fig. 7. We resolved the conflict between nodes 1 and 4 by only allowing node 4 to send the data to node 5 at the second step.

The generalization of bidirectional exchange algorithms is not feasible with this model because of the constraint that in each step, data is sent potentially to N other nodes and received from N nodes, thus saturating the $2N$ links. The cost of our generalized bucket algorithm now becomes

$$(d - N)\alpha + \frac{D_j - 1}{D_j} n\beta$$

where

$$d = \sum_{i=1}^{N} D_i \text{ and } \forall i \neq j, D_j \geq D_i$$

The bandwidth component is factored by $\frac{D_j - 1}{D_j}$ because data needs to be sent only around the ring of the dimension with the most number of nodes. The data needed to be sent along the bucket of all other dimensions is done so simultaneously. The latency is $d - N$ for our algorithm because that many steps are required for any data message to traverse the entire torus.

The bandwidth component not only is at least twice the lower bound, but

$$\frac{p-1}{p} n\beta > \frac{D_j - 1}{D_j} n\beta > \frac{p-1}{p} \frac{n\beta}{N}$$

since at each step data cannot always be sent to N neighboring nodes because of the conflict described above. Despite this conflict our algorithm is an improvement over the original bucket algorithm.

4.2.3 Derived Long-Vector Broadcast

A long-vector broadcast algorithm can be derived by combining a scatter with an allgather operation. The cost becomes

$$T_{LongBcast}(p, n) = (\lceil \log_2(p) \rceil + p - 1)\alpha + 2\frac{p-1}{p} n\beta$$

when using the original one-dimensional algorithms.

By deriving the long-vector broadcast algorithm using the N-dimensional algorithms of MST scatter and bucket allgather, the cost now is

$$(\lceil \log_{2N+1}(p) \rceil + d - N)\alpha + \left(\frac{p-1}{2Np} + \frac{D_j - 1}{D_j} \right) n\beta$$

Clearly we have reduced both the latency and bandwidth cost of the original derived long-vector broadcast.

4.2.4 Summary

The original bucket algorithm was designed specifically to avoid network conflicts, and the generalized bucket algorithm can and should be implemented to avoid any network conflicts. When implementing reduce-scatter with the generalized bucket algorithm, the lower bound, $\frac{p-1}{p} n\gamma$, for the computation component is also achieved.

Other collective operations can also be derived in similar fashion to the broadcast. For example allreduce being implemented with the new generalized MST reduce(-to-one) followed by MST broadcast or by the generalized bucket reduce-scatter followed by bucket allgather [19].

5. Performance

Here we describe the performance of an implementation of our algorithms. The architecture on which the implementations were benchmarked is presented, followed by the results attained from that architecture.

All results are presented in logarithmic graphs. The length of data ranges from 8 B to 4 MB, and time in seconds is the unit of measure. The tests were conducted on 512 nodes within an 8 $\times$ 8 $\times$ 8 torus.

5.1 Testbed Architecture

The assumption that a node can send or receive from $2N$ nodes *simultaneously* is the main departure of this paper from previous work on collective communication. The IBM Blue Gene/L (BG/L) [11] is a new supercomputer architecture that supports this new functionality, but more discussion is needed to evaluate the extent of that claim.

BG/L performs point-to-point communication using a $3D$ torus network. One BG/L rack comprises 1024 dual-processor compute nodes divided into two midplanes. A job requested on BG/L is given a physical $3D$ partition that best fits the number of nodes requested even if the number of nodes does not fill the entire partition. A job can use the torus topology only if it is executed on a midplane or rack. Otherwise smaller partitions are merely meshes.

BG/L also has two modes: coprocessor and virtual node mode. We deal only with coprocessor mode in our experiments where both processors on the node run a single thread. One processor handles computation while the other processor handles cache coherency and helps route data packets among other things.

We use MPI to implement the collective communication operations. The MPI implementation on BG/L [1] is an adaptation of the MPICH2 library from Argonne National Laboratory [24, 26]. Multiple calls to `MPI_Isend` facilitate sending simultaneously [23]. Packets are routed by using either a deterministic or an adaptive routing algorithm on an individual basis. Since we cannot be sure of which routing algorithm is used for any particular packet, one way we try to ensure that a node can send simultaneously to six other nodes is to send only to nodes that lie along the same dimensional axis as the sending node. Sending to two or more nodes that are not along any axis may lead to network conflicts.

5.1.1 Sending Simultaneously

Attempting to send to multiple nodes simultaneously on BG/L does not take the same amount of time as just sending to one other node. We represent the extra time to send simultaneously with τ as a factor of the overall time to send, so the cost to send simultaneously is represented by

$$(\alpha + n\beta)(1 + (l - 1)\tau)$$

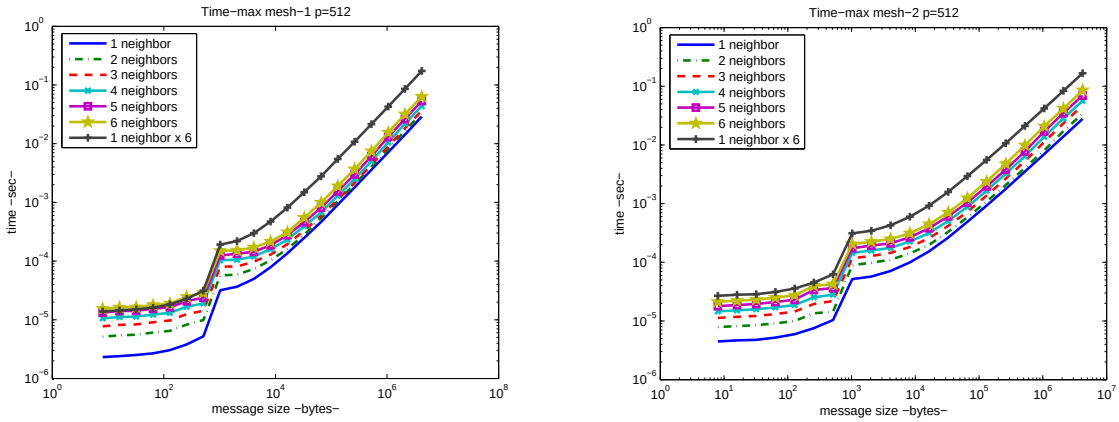

Figure 8. Left: Longest time for unidirectional sending or receiving on a mesh topology. Right: Longest time for bidirectional exchange on a mesh topology.

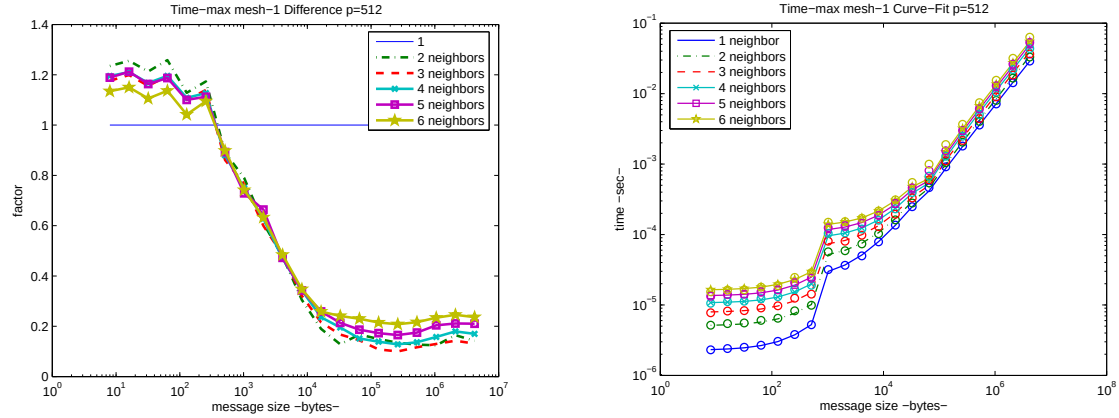

Figure 9. Left: Values of τ for the longest time for unidirectional sending or receiving on a mesh topology. Right: Curve fitting based on the experimental values of α, β, and τ for unidirectional sending or receiving on a mesh topology.

where l is the number of nodes to send, and $0 \leq \tau \leq 1$. If $\tau = 0$, then sending to multiple nodes is free, but if $\tau = 1$, then there is no benefit from performing simultaneous sends.

Values for α, β, and τ can be found with simple benchmarking. The longest, shortest, and average times for any node to send or receive with one to six neighbors simultaneously were gathered. The sends and receives are ordered in such a way that all nodes take part in sending or receiving with a certain number of neighboring nodes. We also time bidirection exchange, where each node sends and receives, as opposed to one or the other, with a different number of neighbors simultaneously. We assume an underlying mesh topology along with the torus in cases where the partition of nodes is smaller than a midplane. This condition is handled by constraining the nodes on the edges of the partition not to send around the ring of a particular dimension.

The results for the longest time to perform a send or receive and bidirectional exchange assuming a mesh topology are presented in Fig. 8. Given all the data gathered from our benchmark, the experimental values of τ are determined on BG/L and are shown in the left of Fig. 9 where the baseline for our comparisons is the time to perform unidirectional sends or receives with only one other node. We performed curve fitting in the right of Fig. 9 for three sets of data lengths: 8 to 512 B, 1 to 32 KB, and 64 KB to 4 MB. The lines in the curve fitting graph represent the cost of communication based on the experimental values of α, β, and τ superimposed onto data circles, representing the original data from the graph in the left of Fig. 8, where we can clearly see a tight relationship.

For the first set of data lengths, 8 to 512 B, $\tau = 0.9797$ whereas $\tau = 0.1792$ for the third set of data lengths, 64 KB to 4 MB with unidirectional sending and receiving on a mesh topology. The first set corresponds to the use of the eager protocol by the send routine. For greater data lengths, the rendezvous protocol is used. The time to send to six neighbors simultaneously is less than six times the time to send to one other node.

We note that bidirectional exchange performed with only one other node is optimized to perform nearly as well as a unidirectional send or receive when using the rendezvous protocol. The values of α and β are relatively close with bidirectional exchange and unidirectional sends and receives, but the value of τ is nearly double, so bidirectional exchange does not scale for simultaneous exchanges. Thus algorithms such as recursive-doubling and halving still pro-

Timings (ms)	Data Lengths		
Implementation	1 MB	2 MB	4 MB
`MPI-Bcast`	37	45	56
`my-bcast-Bucket-old`	39	47	58
`my-bcast-Bucket-new`	36	42	49
`MPI-Scatter`	4.6	8.8	17
`my-scatter-MST-old`	3.7	7.1	14
`my-scatter-MST-new`	3.2	5.6	10
`MPI-Allgather`	250	280	340
`my-allgather-Bucket-old`	35	39	45
`my-allgather-Bucket-new`	33	36	39

Table 2. Timing results for long-vector implementations from 1 to 4 MB in milliseconds.

vide good performance despite not being able to be generalized to send data simultaneously.

5.2 Results

In the graphs in Table 2 and Fig. 10 we report the following curves:

- The lines labeled `MPI` are the default implementations of the operations on the testbed architecture. These are used with comparison to our implementations.
- The lines labeled `MST` are for the implementations of the short-vector algorithms.
- The lines labeled `Bucket` are for the implementations of the long-vector algorithms.
- The lines labeled with `old` are for our original implementations of algorithms that assume a linear array topology. These are also used to compare with our new generalized algorithms.
- The lines labeled with `new` are for the implementations of the new generalized algorithms on a three-dimensional torus.

In Table 3 we present the cost of the overhead in implementing the new generalized and original algorithms. The overhead that mainly consists of calculating indices is denoted by ϕ, which is a constant function of the number of dimensions. For instance eight arrays of length N are used in our implementation of the new generalized MST broadcast. $\phi(3)$ represents the overhead of the implementations of the new generalized algorithms whereas $\phi(1)$ represents the implementations of the original algorithms. These results are found by taking the average time to execute the implementations without performing any communication across all data lengths.

5.2.1 Broadcast

In the left of Fig. 10 we compare implementations of broadcast for short lengths of data, 8 B to 16 KB. Since data messages are not sent simultaneously on BG/L at data lengths less than 1 KB, the new generalized MST algorithm of broadcast nearly emulates the original MST algorithm, but the significant overhead required to implement our new generalized algorithm, $\phi(3)$, derogates its performance compared to the much simpler implementation of the original MST algorithm. Our new implementation gains a factor of two performance around 2 KB which because of the decrease in the number of stages from the original, $\lceil \log_2(p) \rceil$ MST algorithm.

In Table 2 we compare implementations of broadcast for long lengths of data, 1 to 4 MB. Even though our new derived long-vector broadcast does outperform our old long-vector and IBM MPI's implementations, performance gains are bounded by the limitations of the allgather building block.

Timings (μs)	Overhead		
Implementation	$\phi(3)$	$\phi(1)$	Ratio
`my-bcast-MST`	30	0.52	58
`my-scatter-MST`	520	1.3	400
`my-allgather-Bucket`	2300	2.8	821

Table 3. Timing results in microseconds and ratios of the overhead, ϕ, required for the implementations of the new generalized and original algorithms.

5.2.2 Scatter

In Table 2 we can clearly see that our new generalized MST scatter performs quite well for long lengths of data. For short lengths of data, our original and IBM's implementations of scatter perform quite favorably compared to our new generalized scatter because of the overhead, which is a factor of four hundred difference shown in Table 3. Since we use MST scatter as a building block for the long-vector broadcast, this is still a significant performance gain.

5.2.3 Allgather

In Table 2 our new generalized bucket allgather sees only a marginal performance gain for long lengths of data from our original bucket algorithm. This is due to the following factors:

- Since the bucket algorithm was designed for long data lengths, the β term dominates the cost. Our new generalized algorithm decreases the bandwidth cost by only a small fraction from the original bucket algorithm where both β terms asymptotically approach the bound of $\Theta(n)$.
- The significant overhead in the new generalized implementation is over a factor of eight hundred from the original and simpler implementations given in Table 3.
- The bucket algorithm assumes the topology is a full torus, but our tests were conducted on a midplane. A BG/L rack consists of 1024 nodes within a $16 \times 8 \times 8$ torus, so messages in a midplane can be sent around the ring physically in only two of the three possible dimensions. Since our algorithm assumes a full torus, network conflicts will occur because messages are sent around the ring of each dimension at every step.
- Our cost analysis assumes that sending simultaneously up to $2N$ other nodes costs the same as sending to only one other node, but we can clearly see that $\tau \gg 0$ from our benchmark results.

We show the factor of eight performance gain between our implementation of the bucket allgather and IBM's default implementation of allgather in the right of Fig. 10. This clearly shows a glitch in IBM's default implementation of allgather.

5.2.4 Summary

We can obtain performance gains with our new generalized algorithms, but significant factors limit that gain. Even though optimizations can be made to our new implementations, we have shown that a performance gain of N, three on BG/L, is not attainable because of inherent limits in the capabilities of the architecture and the algorithms themselves.

6. Conclusions

Our generalized algorithms make use of the physical underlying N-dimensional torus. We are able to achieve the lower bound for latency using the MST algorithm. Our MST scatter is also able to achieve the lower bound for bandwidth, but our bucket

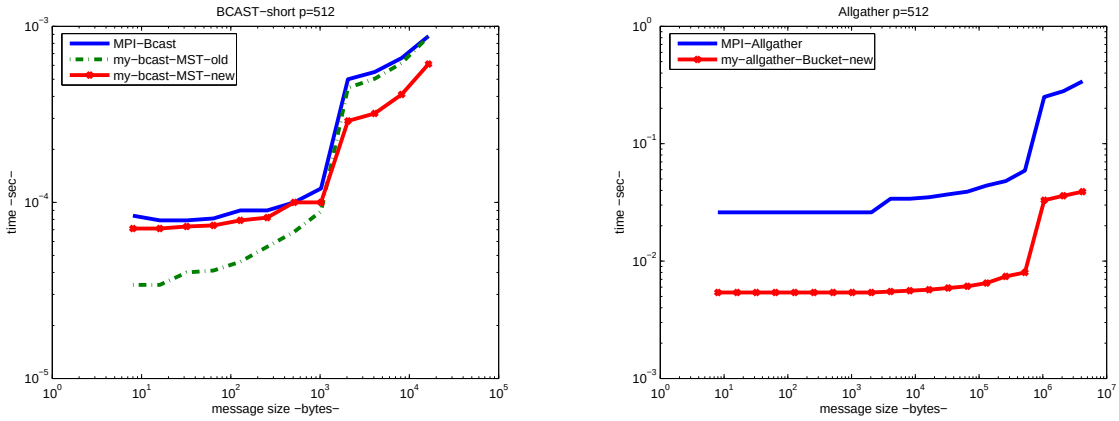

Figure 10. Left: Comparing short-vector implementations of broadcast. Right: Comparing our bucket allgather to IBM's default implementation of allgather.

algorithm, and any possible algorithm, cannot achieve the lower bound bandwidth for allgather because of the constraints of our model of parallel computation. Despite the lower expected costs of these algorithms, the implementations of these algorithms are limited by the physical mechanisms of the system and by the significant overhead needed.

More advanced techniques such as imposing higher-dimensional virtual topologies in order to develop hybrid multidimensional algorithms can be used on top of these simultaneously sending algorithms for lower performance costs [8]. When the mechanism of sending simultaneously becomes decoupled from the physical topology, simplier techniques such as the division of p nodes by $N + 1$ for the MST algorithm can be used without penalty.

As part of the current project, we intend to investigate collective communication on systems with SMP nodes that share more than one processor. The virtual node mode on the IBM Blue Gene/L facilitates the use of both processors in a node. We are also not convinced that our generalized bucket algorithm is the most efficient algorithm, so further study is required to develop an algorithm that strives to obtain the lower bound for bandwidth in our long-vector algorithms.

Acknowledgments

This work was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract W-31-109-ENG-38. We thank Chris Bischof for giving us access to the Sun SMP machines at the University of Aachen.

References

[1] G. Almasi, C. Archer, J. G. Castanos, J. A. Gunnels, C. C. Erway, P. Heidelberger, X. Martorell, J. E. Moreira, K. Pinnow, J. Ratterman, B. D. Steinmacher-Burow, W. Gropp, and B. Toonen. Design and implementation of message-passing services for the Blue Gene/L supercomputer. *IBM J. Res. and Dev.*, 49(2/3):393–406, March/May 2005.

[2] M. Barnett, S. Gupta, D. Payne, L. Shuler, R. A. van de Geijn, and J. Watts. Interprocessor collective communication library (intercom). In *Proceedings of the Scalable High Performance Computing Conference 1994*, 1994.

[3] 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. *J. Parallel Distrib. Comput.*, 24:191–201, 1995.

[4] M. Barnett, D. Payne, and R. van de Geijn. Optimal broadcasting in mesh-connected architectures. Computer Science report TR-91-38, Univ. of Texas, 1991.

[5] M. Barnett, D. Payne, R. van de Geijn, and J. Watts. Broadcasting on meshes with wormhole routing. *J. Parallel Distrib. Comput.*, 35(2):111–122, 1996.

[6] Gregory D. Benson, Cho-Wai Chu, Qing Huang, and Sadik G. Caglar. A comparison of MPICH allgather algorithms on switched networks. In Jack Dongarra, Domenico Laforenza, and Salvatore Orlando, editors, *Recent Advances in Parallel Virtual Machine and Message Passing Interface, 10th European PVM/MPI Users' Group Meeting*, pages 335–343. Lecture Notes in Computer Science 2840, Springer, September 2003.

[7] Massimo Bernaschi, Giulio Iannello, and Mario Lauria. Experimental results about MPI collective communication operations. In *Proceedings of HPCN99*, 1999.

[8] Ernie W. Chan, Marcel F. Heimlich, Avi Purkayastha, and Robert A. van de Geijn. On optimizing collective communication. In *Proceedings of the 2004 IEEE International Conference on Cluster Computing*, pages 145–155, San Diego, CA, 2004. IEEE.

[9] Graham E. Fagg, Sathish S. Vadhiyar, and Jack J. Dongarra. ACCT: Automatic collective communications tuning. In Jack Dongarra, Peter Kacsuk, and Norbert Podhorszki, editors, *Recent Advances in Parallel Virutal Machine and Message Passing Interface*, number 1908 in Springer Lecture Notes in Computer Science, pages 354–361, September 2000.

[10] G. Fox, M. Johnson, G. Lyzenga, S. Otto, J. Salmon, and D. Walker. *Solving Problems on Concurrent Processors*, volume I. Prentice Hall, 1988.

[11] A. Gara, M. A. Blumrich, D. Chen, G. L.-T. Chiu, P. Coteus, M. E. Giampapa, R. A. Haring, P. Heidelberger, D. Hoenicke, G. V. Kopcsay, T. A. Liebsch, M. Ohmacht, B. D. Steinmacher-Burow, T. Takken, and P. Vranas. Overview of the Blue Gene/L system architecture. *IBM J. Res. and Dev.*, 49(2/3):195–212, March/May 2005.

[12] Ching-Tien Ho and S. Lennart Johnsson. Distributed routing algorithms for broadcasting and personalized communication in hypercubes. In *Proceedings of the 1986 International Conference on Parallel Processing*, pages 640–648. IEEE, 1986.

[13] S. L. Johnsson and C. T. Ho. Optimum broadcasting and personalized communication in hypercubes. *IEEE Transactions on Computers*, pages

1249–1268, September 1989.

[14] L. V. Kale, Sameer Kumar, and Krishnan Vardarajan. A framework for collective personalized communication. In *Proceedings of the 17th International Parallel and Distributed Processing Symposium (IPDPS '03)*, 2003.

[15] N. Karonis, B. de Supinski, I. Foster, W. Gropp, E. Lusk, and J. Bresnahan. Exploiting hierarchy in parallel computer networks to optimize collective operation performance. In *Proceedings of the Fourteenth International Parallel and Distributed Processing Symposium (IPDPS '00)*, pages 377–384, 2000.

[16] T. Kielmann, R.F.H. Hofman, H.E. Bal, A. Plaat, and R.A.F. Bhoedjang. MagPIe: MPI's collective communication operations for clustered wide area systems. In *ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP'99)*, pages 131–140. ACM, May 1999.

[17] S.L. Lillevik. The Touchstone 30 Gigaflop DELTA Prototype. In *Sixth Distributed Memory Computing Conference Proceedings*, pages 671–677. IEEE Computer Society Press, 1991.

[18] Prasenjit Mitra, David Payne, Lance Shuler, Robert van de Geijn, and Jerrell Watts. Fast collective communication libraries, please. In *Proceedings of the Intel Supercomputing Users' Group Meeting 1995*, 1995.

[19] D. Payne, L. Shuler, R. van de Geijn, and J. Watts. Streetguide to collective communication. unpublished manuscript.

[20] Rolf Rabenseifner and Gerhard Wellein. Communication and optimization aspects of parallel programming models on hybrid architectures. *International Journal of High-Performance Computing Applications*, 17(1):49–62, 2003.

[21] Y. Saad and M.H. Schultz. Data communications in hypercubes. *J. Parallel Distrib. Comput.*, 6:115–135, 1989.

[22] Mohak Shroff and Robert A. van de Geijn. Collmark MPI collective communication benchmark. unpublished manuscript, 2001.

[23] Marc Snir, Steve Otto, Steven Huss-Lederman, David W. Walker, and Jack Dongarra. *MPI: The Complete Reference*, volume 1, The MPI Core. The MIT Press, 2nd edition, 1998.

[24] Rajeev Thakur and William Gropp. Improving the performance of collective operations in MPICH. In *Proceedings of the 10th European PVM/MPI Users' Group Conference (Euro PVN/MPI 2003)*, pages 257–267, September 2003.

[25] Rajeev Thakur, William Gropp, and Brian Toonen. Optimizing the synchronization operations in MPI one-sided communication. *International Journal of High-Performance Computing Applications*, 19(2):119–128, Summer 2005.

[26] Rajeev Thakur, Rolf Rabenseifner, and William Gropp. Optimization of collective communication operations in MPICH. *International Journal of High-Performance Computing Applications*, (19)1:49–66, Spring 2005.

[27] V. Tipparaju, J. Nieplocha, and D. K. Panda. Fast collective operations using shared and remote memory access protocols on clusters. In *Proceedings of the 17th International Parallel and Distributed Processing Symposium (IPDPS '03)*, 2003.

[28] Jesper Larason Traff and Andreas Ripke. An optimal broadcast algorithm adapted to SMP clusters. In *EuroPVM/MPI 2005, LNCS 3666*, pages 48–56, 2005.

[29] Jesper Larason Traff and Andreas Ripke. Optimal broadcast for fully connected networks. In *HPCC 2005, LNCS 3726*, pages 45–56, 2005.

[30] Sathish S. Vadhiyar, Graham E. Fagg, and Jack Dongarra. Automatically tuned collective communication. In *Proceedings of Supercomputing 2000*, Dallas, TX.

[31] Robert van de Geijn. On global combine operations. *J. Parallel Distrib. Comput.*, 22:324–328, 1994.

[32] Jerrell Watts and Robert van de Geijn. A pipelined broadcast for multidimensional meshes. *Parallel Processing Letters*, 5(2):281–292, 1995.

[33] Thomas Worsch, Ralf Reussner, and Werner Augustin. On benchmarking collective MPI operations. In Dieter Kranzlmüller, Peter Kacsuk, Jack Dongarra, and Jens Volkert, editors, *Recent Advances in Parallel Virtual Machine and Message Passing Interface, 9th European PVM/MPI Users' Group Meeting*, pages 271–279. Lecture Notes in Computer Science 2474, Springer, September 2002.