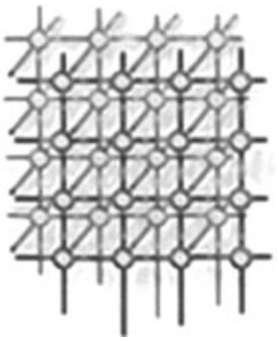
CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2007; **19**:1749–1783
Published online 5 July 2007 in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/cpe.1206



Collective communication: theory, practice, and experience

Ernie Chan[1,*,†], Marcel Heimlich[1], Avi Purkayastha[2] and Robert van de Geijn[1]

[1]*Department of Computer Sciences, The University of Texas at Austin, Austin, TX 78712, U.S.A.*
[2]*Texas Advanced Computing Center, The University of Texas at Austin, Austin, TX 78712, U.S.A.*

SUMMARY

We discuss the design and high-performance implementation of collective communications operations on distributed-memory computer architectures. Using a combination of known techniques (many of which were first proposed in the 1980s and early 1990s) along with careful exploitation of communication modes supported by MPI, we have developed implementations that have improved performance in most situations compared to those currently supported by public domain implementations of MPI such as MPICH. Performance results from a large Intel Xeon/Pentium 4 (R) processor cluster are included.

Received 14 September 2006; Revised 24 January 2007; Accepted 10 March 2007

KEY WORDS: collective communication; distributed-memory architecture; clusters

1. INTRODUCTION

This paper makes a number of contributions to the topic of collective communication:

1. *A review of best practices*: Collective communication was an active research in the 1980s and early 1990s as distributed-memory architectures with large numbers of processors were first introduced [1–7]. Since then an occasional paper has been published [8–16], but no dramatic new developments have been reported.

*Correspondence to: Ernie Chan, Department of Computer Sciences, The University of Texas at Austin, Austin, TX 78712, U.S.A.
†E-mail: echan@cs.utexas.edu

Contract/grant sponsor: National Science Foundation; contract/grant number: CCF-0540926

2. *Focus on simplicity*: Historically, hypercube topologies were the first topologies used for distributed-memory architectures. Examples include Caltech's Cosmic Cube [17], the Intel iPSC, NCUBE [18], and Thinking Machines' Connection Machine architectures. As a result, highly efficient algorithms for hypercubes were developed first, and these algorithms were then modified to target architectures with alternative topologies. Similarly, textbooks often start their discussion of collective communication by considering hypercubes.
   In our exposition, we take a different approach by considering one-dimensional topologies first. Algorithms that perform well are then generalized to multidimensional meshes. Hypercubes are finally discussed briefly by observing that they are $\log(p)$ dimensional meshes with two (computational) nodes in each dimension. This fact allows us to focus on simple, effective solutions that naturally generalize to higher dimensions and ultimately hypercubes.
3. *Algorithms*: One consequence of Item 2 is that we can state the algorithms more simply and concisely. Minimum-spanning tree algorithms on hypercubes typically required loop-based algorithms that computed indices of destination nodes by 'toggling' bits of the indices of source nodes. We instead present the algorithms recursively and avoid such obscuring by restricting bit manipulation.
4. *Analysis*: The cost of algorithms is analyzed *via* a simple but effective model of parallel computation.
5. *Tunable libraries*: More recently, the topic of tuning, preferably automatically, of collective communication libraries has again become fashionable [19]. Unfortunately, many of these papers focus on the mechanism for choosing algorithms from a loose collection of algorithms. Often this collection does not even include the fastest and/or most practical algorithm. *Perhaps the most important contribution of this paper is that it shows how algorithms for a given operation can be organized as a parameterized family*, which then clearly defines what parameters can be tuned to improve performance. This approach was already incorporated into the highly tuned InterCom library for the Intel Touchstone Delta and Paragon architectures of the early and mid-1990s [20–22]. However, many details of the theory and practical techniques used to build that library were never published.
6. *Implementation*: The merits of the approach are verified *via* a MPI-compatible implementation of all the presented algorithms [23]. Experiments show that the resulting implementation is comparable and sometimes better than the MPICH implementation of the Message-Passing Interface (MPI) [24–26].

There is an entire class of algorithms that we do not treat: pipelined algorithms [2,3,27]. The reason is that we do not consider these practical on current generation architectures.

The remainder of the paper is organized as follows. In Section 2, we explain some basic assumptions that are made for the purpose of presenting this paper. In Section 3, we discuss the communication operations. Section 4 delineates the lower bounds of the collective communication operations followed by a discussion of network topologies in Section 5. In Section 6, we discuss different algorithms for varying data lengths. In Section 7, we discuss strategies for the special cases where short and long vectors of data are communicated. More sophisticated hybrid algorithms that combine techniques for all vector lengths are discussed in Section 8. Performance results are given in Section 9. Concluding remarks can be found in Section 10.

DOI: 10.1002/cpe

2. MODEL OF PARALLEL COMPUTATION

To analyze the cost of the presented algorithms, it is useful to assume a simple model of parallel computation. These assumptions are:

- *Target architectures*: Currently, the target architectures are distributed-memory parallel architectures. However, we expect that the methods discussed in this paper also have applicability when many cores on a single processor become available.
- *Indexing*: This paper assumes a parallel architecture with p computational nodes (nodes hereafter). The nodes are indexed from 0 to $p - 1$. Each node could consist of a Symmetric Multi-Processor (SMP) but for communication purposes will be treated as one unit.
- *Logically fully connected*: We will assume that any node can send directly to any other node through a communication network where some topology provides automatic routing.
- *Communicating between nodes*: At any given time, a single node can send only one message to one other node. Similarly, it can only receive one message from one other node. We will assume a node can send and receive simultaneously.
- *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 startup time and per data item transmission time.
  In our model, the cost of the message is *not* a function of the distance between two nodes. The start-up cost is largely due to software overhead on the sending and the receiving nodes. The routing of messages between nodes is subsequently done in hardware using *wormhole routing*, which pipelines messages and incurs a very small extra overhead due to the distance between two nodes [17]. Typically, α is four to five orders of magnitude greater than β where β is on the order of the cost of an instruction.
- *Network conflicts*: Assuming that the path between two communicating nodes, determined by the topology and the routing algorithm, is completely occupied, then if some link (connection between neighboring nodes) in the communication path is occupied by two or more messages, a network conflict occurs. This extra cost is modeled with $\alpha+kn\beta$ where k is the maximum over all links (along the path of the message) of the number of conflicts on the links. Alternatively, links on which there is a conflict may be multiplexed, which yields the same modification to the cost.
- *Bidirectional channels*: We will assume that messages traveling in opposite directions on a link do not conflict.
- *Cost of computation*: The cost required to perform an arithmetic operation (e.g. a reduction operation) is denoted by γ.

Although simple, the above assumptions are useful when conducting an analysis of communication costs on actual architectures.

Some additional discussion is necessary regarding parameters α and β:

- *Communication protocols*: The most generic communication uses the so-called three-pass protocol. A message is sent to alert the receiving node that a message of a given size will be sent. After the buffer space for the message has been allocated, the receiving node responds.

DOI: 10.1002/cpe

Finally, the message itself is sent. Note that this requires three control messages to be sent between the sending and receiving nodes. We will denote the latency associated with this protocol by α_3.

If the sender can rely on the fact that a receive buffer already exists, a one-pass protocol can be used, in which the message is simply sent without the above-described handshake. We will denote the latency associated with this protocol by α_1. In particular, we will assume that there is always static buffer space for very short messages.

The three-pass protocol can easily cost up to three times more than the one-pass protocol. Thus, in our discussion we will assume that $\alpha_3 = 3\alpha_1$.

- *Relative order of send and receive calls*: The cost per item sent is affected by the relative order in which a send and corresponding receive are posted. If a send is initiated before the corresponding receive is posted on the receiving node, the incoming message is buffered in temporary space and copied when the receive, which indicates where the message is to be finally stored, is posted. If, on the other hand, the receive is posted before the send, or the send blocks until the receive is posted, no such extra copy is required. We will denote the cost per item transferred by β_1 if no extra copy is required and by β_2 if it is.

3. COLLECTIVE COMMUNICATION

When the nodes of a distributed-memory architecture collaborate to solve a given problem, inherently computation previously performed on a single node is now distributed among the nodes. Communication is performed when data are shared and/or contributions from different nodes must be consolidated. Communication operations that simultaneously involve a group of nodes are called *collective communication* operations. In our discussions, we will assume that the group includes all nodes.

The most typically encountered collective communications, discussed in this section, fall into two categories:

- *Data redistribution operations*: Broadcast, scatter, gather, and allgather. These operations move data between processors.
- *Data consolidation operations*: Reduce(-to-one), reduce–scatter, and allreduce. These operations consolidate contributions from different processors by applying a reduction operation. We will only consider reduction operations that are both commutative and associative.

The operations discussed in this paper are illustrated in Figure 1. In that figure, x indicates a vector of data of length n. For some operations, x is subdivided into subvectors x_i, $i = 0, \ldots, p-1$, where p equals the number of nodes. A superscript is used to indicate a vector that must be reduced with other vectors from other nodes. $\sum_j x^{(j)}$ indicates the result of that reduction. The summation sign is used because summation is the most commonly encountered reduction operation.

We present these collective communications as pairs of *dual* operation. We will show later that an implementation of an operation can be transformed into that of its dual by reversing the communication (and adding to or deleting reduction operations from the implementation). These

DOI: 10.1002/cpe

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

Figure 1. Collective communications considered in this paper.

dual pairs are indicated by the groupings in Figure 1 (separated by the thick lines): broadcast and reduce(-to-one), scatter and gather, and allgather and reduce–scatter. Allreduce is the only operation that does not have a dual (or it can be viewed as its own dual).

DOI: 10.1002/cpe

4. LOWER BOUNDS

It is useful to present lower bounds on the cost of these operations under our model regardless of the implementation. In this section, we give informal arguments to derive these lower bounds. We will treat three terms of communication cost separately: latency, bandwidth, and computation. Lower bounds are summarized in Table I. It is assumed that $p>1$ and that subvectors x_i and $x_i^{(j)}$ have equal length.

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. Under our model, at each step, we can at most double the number of nodes that get the data.

Computation: Only the reduction communications 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 perfectly among the nodes reduces the time to $((p-1)/p)n\gamma$ 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 is bounded below by $n\beta$. For the gather and scatter, the root node must either send or receive $((p-1)/p)n$ items, with a cost of at least $((p-1)/p)n\beta$. The same is the case for all nodes during the allgather and reduce–scatter. The allreduce is somewhat more complicated. If the lower bound on computation is to be achieved, one can argue that $((p-1)/p)n$ items must leave each node, and $((p-1)/p)n$ items must be received by each node after the computation is completed for a total cost of at least $2((p-1)/p)n\beta$. For further details see [21].

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

Communication	Latency	Bandwidth	Computation
Broadcast	$\lceil \log_2(p) \rceil \alpha$	$n\beta$	—
Reduce(-to-one)	$\lceil \log_2(p) \rceil \alpha$	$n\beta$	$\frac{p-1}{p} n\gamma$
Scatter	$\lceil \log_2(p) \rceil \alpha$	$\frac{p-1}{p} n\beta$	—
Gather	$\lceil \log_2(p) \rceil \alpha$	$\frac{p-1}{p} n\beta$	—
Allgather	$\lceil \log_2(p) \rceil \alpha$	$\frac{p-1}{p} n\beta$	—
Reduce-scatter	$\lceil \log_2(p) \rceil \alpha$	$\frac{p-1}{p} n\beta$	$\frac{p-1}{p} n\gamma$
Allreduce	$\lceil \log_2(p) \rceil \alpha$	$2\frac{p-1}{p} n\beta$	$\frac{p-1}{p} n\gamma$

Note: Pay particular attention to the conditions for the lower bounds given in the text.

DOI: 10.1002/cpe

5. TOPOLOGIES

In this section, we discuss a few topologies. The topology with least connectivity is the linear array. A fully connected network is on the other end of the spectrum. In between, we consider higher dimensional mesh topologies. Hypercubes, which have historical and theoretical value, are shown to be mesh architectures of dimension $\log(p)$ with two nodes in each dimension. Many current architectures have multiple levels of switches that route messages between subnetworks that are typically fully connected.

Linear arrays: We will assume that the nodes of a linear array are connected so that node i has neighbors $i-1$ (left) and $i+1$ (right), $1 \leqslant i < p-1$. Nodes 0 and $p-1$ do not have left and right neighbors, respectively.

A ring architecture would connect node 0 with $p-1$. However, the communication that a ring facilitates is important to us (e.g. simultaneous sending by all nodes to their right neighbor), yet this communication can be achieved on a linear array because the message from node $p-1$ to node 0 does not conflict with any of the other messages under our model.

Mesh architectures: The nodes of a mesh architecture of dimension D can be indexed using a D-tuple, $(i_0, \ldots, i_{D-1})$, with $0 \leqslant i_j < d_j$ and $p = d_0 \times \cdots \times d_{D-1}$. The nodes indexed by $(i_0, \ldots, i_{j-1}, k, i_{j+1}, \ldots, i_{D-1})$, $0 \leqslant k < d_j$, form a linear array.

A torus architecture is the natural extension of a ring to multiple dimensions. We will not need tori for our discussion for the same reason that we do not need rings.

Hypercubes: In Figure 2 we show how a hypercube can be inductively constructed:

- A hypercube of dimension 0 consists of a single node. No bits are required to index this one node.
- A hypercube of dimension d is constructed from two hypercubes of dimension $d-1$ by connecting nodes with corresponding index, and adding a leading binary bit to the index of each node. For all nodes in one of the two hypercubes this leading bit is set to 0 while it is set to 1 for the nodes in the other hypercube.

Some observations are:

- Two nodes are neighbors if and only if the binary representation of their indices differ in exactly one bit.
- View the nodes of a hypercube as a linear array with nodes indexed $0, \ldots, p-1$. Then,
  - the subsets of nodes $\{0, \ldots, p/2-1\}$ and $\{p/2, \ldots, p-1\}$ each form a hypercube; and
  - node i of the left subset is a neighbor of node $i + p/2$ in the right subset.

  This observation applies recursively to the two subsets.

A hypercube is a mesh of dimension $\log(p)$ with two nodes in each dimension.

Fully connected architectures: In a fully connected architecture, all nodes are neighbors of all other nodes. We will see in the remainder of this paper that the primary advantage of a fully connected architecture is that one can view such architectures as higher dimensional meshes by factoring the number of nodes, p, into integer factors.

DOI: 10.1002/cpe

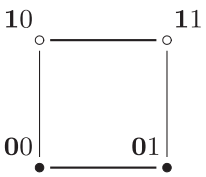
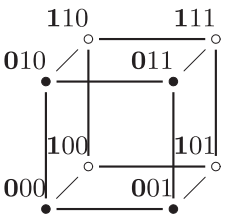
d	p	comment	shape
0	1	A single node.	
1	2	Duplicate the 0 dimensional cube, and connect corresponding nodes with a link.	
2	2^2	Duplicate the 1 dimensional cube, and connect corresponding nodes with a link.	
3	2^3	Duplicate the 2 dimensional cube, and connect corresponding nodes with a link.	
⋮	⋮	⋮	⋮
d	2^d	Duplicate the $d-1$ dimensional cube, and connect corresponding nodes with a link.	⋮

Figure 2. Construction of hypercubes.

6. COMMONLY USED 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 it is the cost of initiating messages, α, that tends to dominate, and algorithms should strive to reduce this cost. In other words, it is the lower bound on the latency in Table I that becomes the dominating factor. When the amount of data is large it is the costs per item sent and/or computed, β and/or γ, that become the dominating factors. In this case, the lower bounds due to bandwidth and computation in Table I are the dominating factors.

6.1. Broadcast and reduce

6.1.1. Minimum-spanning tree algorithms

The best-known broadcast algorithm is the minimum-spanning tree algorithm (MST BCAST). On an arbitrary number of nodes, this algorithm can be described as follows. The nodes $\{0, \ldots, p-1\}$

DOI: 10.1002/cpe

```
MSTBcast( x, root, left, right )

if left = right return
mid = ⌊(left + right)/2⌋
if root ≤ mid then dest = right else dest = left

if me == root Send( x, dest )
if me == dest Recv( x, root )

if me ≤ mid and root ≤ mid
    MSTBcast( x, root, left, mid )
else if me ≤ mid and root > mid
    MSTBcast( x, dest, left, mid )
else if me > mid and root ≤ mid
    MSTBcast( x, dest, mid+1, right )
else if me > mid and root > mid
    MSTBcast( x, root, mid+1, right )
```

(a)

```
MSTReduce( x, root, left, right )

if left = right return
mid = ⌊(left + right)/2⌋
if root ≤ mid then srce = right else srce = left

if me ≤ mid and root ≤ mid
    MSTReduce( x, root, left, mid )
else if me ≤ mid and root > mid
    MSTReduce( x, srce, left, mid )
else if me > mid and root ≤ mid
    MSTReduce( x, srce, mid+1, right )
else if me > mid and root > mid
    MSTReduce( x, root, mid+1, right )

if me == srce Send( x, root )
if me == root Recv( tmp, srce ) and x = x + tmp
```

(b)

```
MSTScatter( x, root, left, right )

if left = right return
mid = ⌊(left + right)/2⌋
if root ≤ mid then dest = right else dest = left

if root ≤ mid
    if me == root Send( x_{mid+1:right}, dest )
    if me == dest Recv( x_{mid+1:right}, root )
else
    if me == root Send( x_{left:mid}, dest )
    if me == dest Recv( x_{left:mid}, root )

if me ≤ mid and root ≤ mid
    MSTScatter( x, root, left, mid )
else if me ≤ mid and root > mid
    MSTScatter( x, dest, left, mid )
else if me > mid and root ≤ mid
    MSTScatter( x, dest, mid+1, right )
else if me > mid and root > mid
    MSTScatter( x, root, mid+1, right )
```

(c)

```
MSTGather( x, root, left, right )

if left = right return
mid = ⌊(left + right)/2⌋
if root ≤ mid then srce = right else srce = left

if me ≤ mid and root ≤ mid
    MSTGather( x, root, left, mid )
else if me ≤ mid and root > mid
    MSTGather( x, srce, left, mid )
else if me > mid and root ≤ mid
    MSTGather( x, srce, mid+1, right )
else if me > mid and root > mid
    MSTGather( x, root, mid+1, right )

if root ≤ mid
    if me == srce Send( x_{mid+1:right}, root )
    if me == root Recv( x_{mid+1:right}, srce )
else
    if me == srce Send( x_{left:mid}, root )
    if me == root Recv( x_{left:mid}, srce )
```

(d)

Figure 3. Minimum-spanning tree algorithms.

are partitioned into two (almost equal) disjoint subsets, $\{0, \ldots, m\}$ and $\{m+1, \ldots, p-1\}$, where $m = \lfloor p/2 \rfloor$ is the 'middle' node. A destination node is chosen in the subset that does not contain the root. The message is sent from the root to the destination after which the root and the destination become the roots for broadcasts within their respective subsets of nodes. This algorithm is given in Figure 3(a). In this algorithm, `x` is the vector data to be communicated, `me` and `root` indicate the index of the node that participates and the current root of the broadcast, and `left` and `right`

DOI: 10.1002/cpe

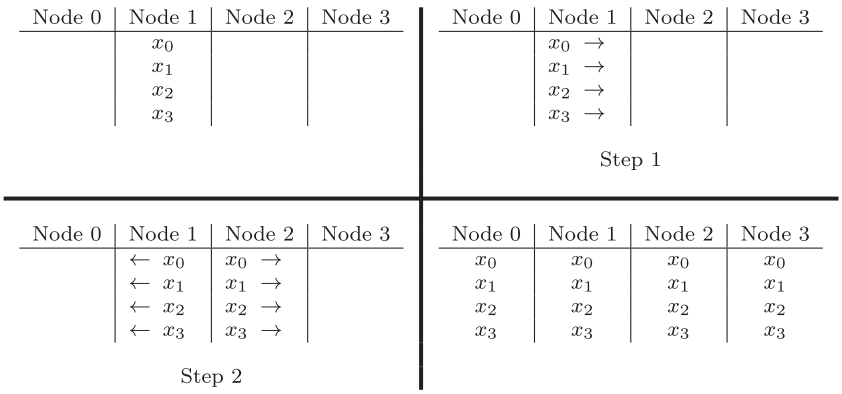


Figure 4. Minimum-spanning tree algorithm for broadcast.

indicate the indices of the left- and right-most nodes in the current subset of nodes. The broadcast among all nodes is then accomplished by calling MSTBCAST(x, `root`, 0, $p-1$). The algorithm is illustrated in Figure 4.

It is not hard to see that, in the absence of network conflicts, the cost of this algorithm is

$$T_{\text{MSTBCAST}}(p,n) = \lceil \log(p) \rceil (\alpha_3 + n\beta_1)$$

in the generic case when the SEND and RECV routines use a three-pass protocol. This cost achieves the lower bound for the latency component of the cost of communication.

Under our model, the algorithm does not incur network conflicts on fully connected networks and on linear arrays, regardless of how the destination is chosen at each step. (The choice of `dest` in Figure 3(a) is simply convenient.) On a hypercube, the destination needs to be chosen to be a neighbor of the current root. This change requires the algorithm in Figure 3(a) to be modified by choosing `dest` as

```
if root≤mid
   dest = root − left + (mid + 1)
else
   dest = root + left − (mid + 1)
```

in other words, choose the node in the subset that does not contain the current root that is in the same relative position as the root.

The MST REDUCE can be implemented by reversing the communication and applying a reduction operation with the data that is received. Again, the nodes $\{0, \ldots, p-1\}$ are partitioned into two (almost equal) disjoint subsets, $\{0, \ldots, m\}$ and $\{m+1, \ldots, p-1\}$. This time a source node is chosen in the subset that does not contain the root. Recursively, all contributions within each subset are reduced to the root and to the source node. Finally, the reduced result is sent from the source node to the root where it is reduced with the data that is already at the root node. This algorithm is given in Figure 3(b) and illustrated in Figure 5.

Comparing the algorithms in Figure 3(a) and (b), we note that the partitioning into subsets of nodes is identical. For the broadcast, the message is sent by the root after which the broadcast

DOI: 10.1002/cpe

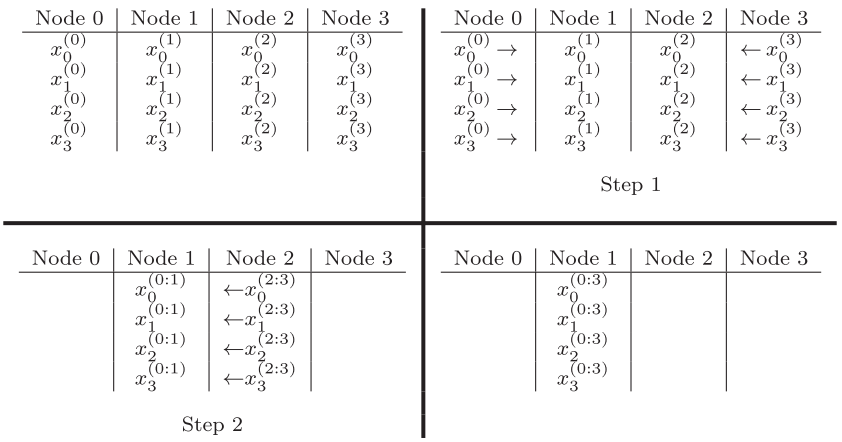


Figure 5. Minimum-spanning tree algorithm for reduce. Notation: $x_i^{(j_0:j_1)} = \sum_j x_i^{(j)}$ where $j \in \{j_0, j_0{+}1, \ldots, j_1\}$.

proceeds recursively in each subset of nodes. For the reduce, the recursion comes first after which a message is sent to the root where the data are reduced with the data at the root. In effect, the communication is reversed in order and direction.

The cost of this algorithm, identical to that of the broadcast except that now a γ term must be added for the reduction at each step, is given by

$$T_{\text{MSTREDUCE}}(p, n) = \lceil \log(p) \rceil (\alpha_3 + n\beta_1 + n\gamma)$$

Both algorithms achieve the lower bound of $\lceil \log(p) \rceil \alpha$ for the latency component of the cost.

6.2. Scatter and gather

A scatter can be implemented much like MST BCAST, except that at each step of the recursion only the data that ultimately must reside in the subnetwork, at which the destination is a member, need to be sent from the root to the destination. The resulting algorithm is given in Figure 3(c) and is illustrated in Figure 6. The MST GATHER is similarly obtained by reversing the communications in the MST SCATTER, as given in Figure 3(d).

Under the assumption that all subvectors are of equal length, the cost of these algorithms is given by

$$T_{\text{MSTSCATTER}}(p, n) = T_{\text{MSTGATHER}}(p, n) = \sum_{k=1}^{\lceil \log(p) \rceil} (\alpha_3 + 2^{-k} n\beta_1) = \lceil \log(p) \rceil \alpha_3 + \frac{p-1}{p} n\beta_1$$

This cost achieves the lower bound for the latency and bandwidth components. Under the stated assumptions, these algorithms are optimal.

DOI: 10.1002/cpe

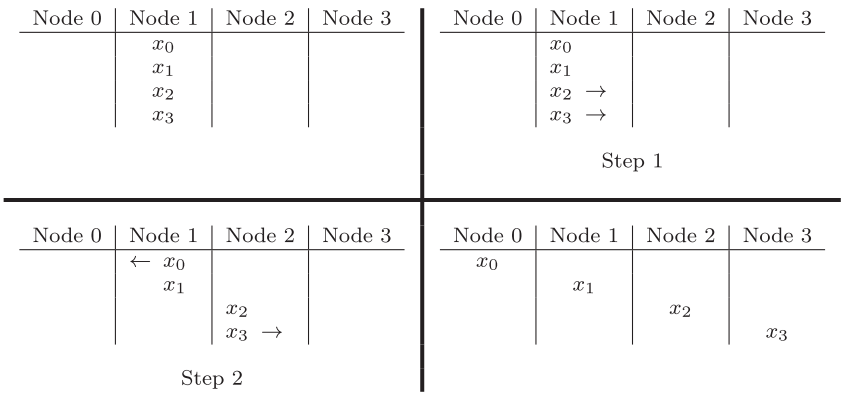


Figure 6. Minimum-spanning tree algorithm for scatter.

6.3. Allgather and reduce–scatter

6.3.1. Bidirectional exchange algorithms

The best known algorithm for allgather assumes that $p = 2^d$ for some integer d and use so-called bidirectional exchanges (BDE), which can be described as follows. Partition the network in two halves. Recursively perform an allgather of all the data in the respective halves. Next, exchange the so-gathered data between disjoint pairs of nodes where each pair consists of one node from each of the two halves. Generally, node i $(i < p/2)$ is paired with node $i + p/2$, which are neighbors if the network is a hypercube. This algorithm, called recursive doubling, is given in Figure 7(a) and illustrated in Figure 8. In the absence of network conflicts (on a hypercube or fully connected architecture) and assuming all subvectors are of equal length, the cost is

$$T_{\text{BDEALLGATHER}}(p, n) = \sum_{k=1}^{\log(p)} (\alpha_3 + 2^{-k} n\beta_1) = \log(p)\alpha_3 + \frac{p-1}{p} n\beta_1$$

This cost attains the lower bound for both the latency and bandwidth components and is thus optimal under these assumptions.

Problems arise with BDE algorithms when the number of nodes is not a power of two. If the subnetwork of nodes contains an odd number of nodes, one "odd" node does not contain a corresponding node in the other subnetwork. In one remedy for this situation, one node from the opposing subnetwork must send its data to the odd node. Unfortunately, this solution requires that one node must send data twice at each step, so the cost of BDE algorithms *doubles* when not using a power of two number of nodes. In practice, BDE algorithms still perform quite well because the node needing to send data twice is different at each step of recursion, so the communication can be overlapped between steps. Nonetheless, the result is rather haphazard.

Reduce–scatter can again be implemented by reversing the communications and their order and by adding reduction operations when the data arrive. This algorithm, called recursive halving, is given in Figure 7(b) and illustrated in Figure 9. Again assuming all subvectors are of equal length,

DOI: 10.1002/cpe

```
BDEAllgather( x, left, right )

if left = right return
size = right − left + 1
mid = ⌊(left + right)/2⌋
if me ≤ mid
    partner = me + ⌊size/2⌋
else
    partner = me − ⌊size/2⌋

if me ≤ mid
    BDEAllgather( x, left, mid )
else
    BDEAllgather( x, mid+1, right )

if me ≤ mid
    Send( x_left:mid, partner )
    Recv( x_mid+1:right, partner )
else
    Send( x_mid+1:right, partner )
    Recv( x_left:mid, partner )
```

(a)

```
BDEReduce-scatter( x, left, right )

if left = right return
size = right − left + 1
mid = ⌊(left + right)/2⌋
if me ≤ mid
    partner = me + ⌊size/2⌋
else
    partner = me − ⌊size/2⌋

if me ≤ mid
    Send( x_mid+1:right, partner )
    Recv( tmp, partner )
    x_left:mid = x_left:mid + tmp
else
    Send( x_left:mid, partner )
    Recv( tmp, partner )
    x_mid+1:right = x_mid+1:right + tmp

if me ≤ mid
    BDEReduce-scatter( x, left, mid )
else
    BDEReduce-scatter( x, mid+1, right )
```

(b)

Figure 7. Bidirectional exchange algorithms for allgather and reduce–scatter.

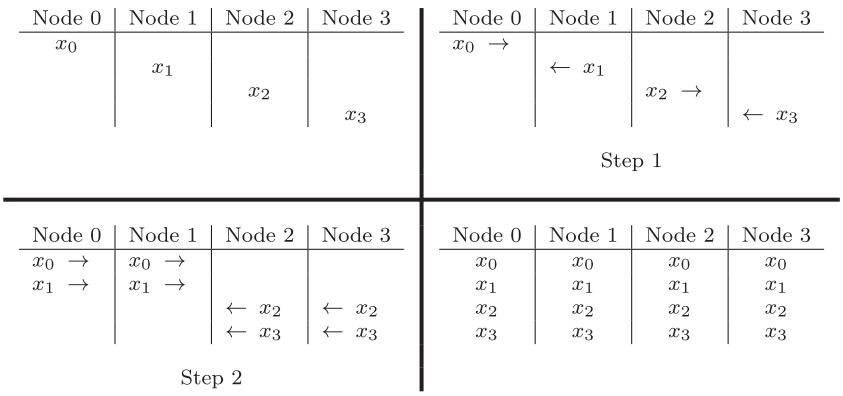


Figure 8. Recursive-doubling algorithm for allgather. In Step 2, bidirectional exchanges occur between the two pair of nodes 0 and 2, and 1 and 3.

the cost is

$$T_{\text{BDEREDUCE–SCATTER}}(p,n) = \sum_{k=1}^{\log(p)} (\alpha_3 + 2^{-k} n(\beta_1 + \gamma)) = \log(p)\alpha_3 + \frac{p-1}{p} n(\beta_1 + \gamma)$$

We will revisit BDE algorithms as a special case of the bucket algorithms, discussed next, on hypercubes in Section 7.3.

DOI: 10.1002/cpe

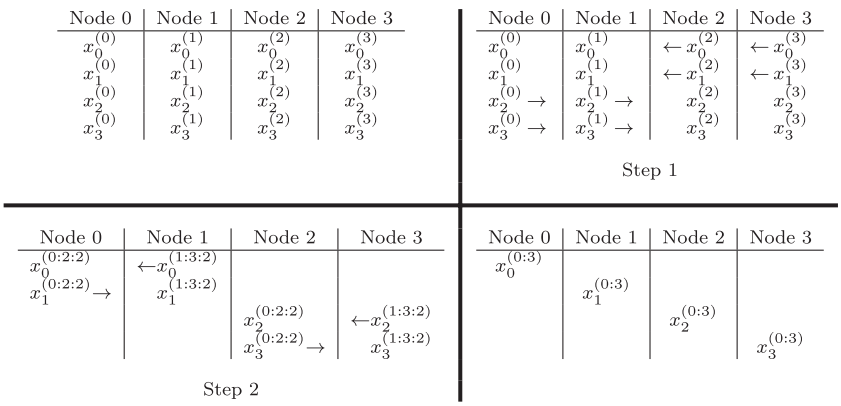


Figure 9. Recursive-doubling algorithm for reduce–scater. In Step 1, bidirectional exchanges occur between the two pair of nodes 0 and 2, and 1 and 3. Notation: $x_i^{(j_0:j_1:s)} = \sum_j x_i^{(j)}$ where $j \in \{j_0, j_0 + s, \ldots, j_1\}$.

```
BKTAllgather( x )

prev = me − 1
if prev < 0 then prev = p − 1
next = me + 1
if next = p then next= 0

curi = me
for i = 0,··· , p − 2
    Send( x_curi, next )
    curi = curi − 1
    if curi < 0 then curi = p − 1
    Recv( x_curi, prev )
endfor
```

(a)

```
BKTReduce-scatter( x )

prev = me − 1
if prev < 0 then prev = p − 1
next = me + 1
if next = p then next = 0

curi = next
for i = p − 2,··· ,0
    Send( x_curi, prev )
    curi = curi + 1
    if curi = p then curi = 0
    Recv( tmp, next )
    x_curi = x_curi + tmp
endfor
```

(b)

Figure 10. Bucket algorithms for allgather and reduce–scatter.

6.3.2. Bucket algorithms

An alternative approach to the implementation of the allgather operation views the nodes as a ring, embedded in a linear array by taking advantage of the fact that messages traversing a link in opposite direction do not conflict. 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 bucket (BKT) algorithm, also called the cyclic algorithm, is given in Figure 10(a) and is illustrated in Figure 11. Note that if each node starts with an equal subvector of data the cost of this approach is given by

$$T_{\text{BKTALLGATHER}}(p, n) = (p-1)\left(\alpha_3 + \frac{n}{p}\beta_1\right) = (p-1)\alpha_3 + \frac{p-1}{p}n\beta_1$$

achieving the lower bound for the bandwidth component of the cost.

DOI: 10.1002/cpe

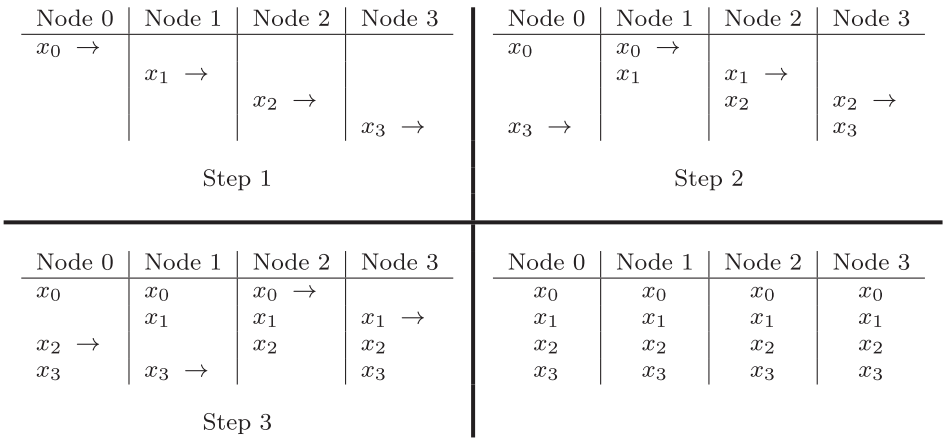


Figure 11. Bucket algorithms for allgather.

A simple optimization comes from preposting all receives after which a single synchronization with the node to the left (indicating that all receives have been posted) is required before sending commences so that a one-pass protocol can be used. This synchronization itself can be implemented by sending a zero-length message, at a cost of α_1 in our model.[‡] The remaining sends each also incur only α_1 as a latency cost, for a total cost of

$$T_{\text{BKTALLGATHER}}(p,n) = \alpha_1 + (p-1)\alpha_1 + \frac{p-1}{p}\,n\beta_1 = p\alpha_1 + \frac{p-1}{p}\,n\beta_1$$

Similarly, the reduce–scatter operation can be implemented by a simultaneous passing of messages around the ring to the left. The algorithm is given in Figure 10(b) and is illustrated in Figure 12. This time, a partial result towards the total reduction of the subvectors are accumulated as the messages pass around the ring. With a similar strategy for preposting messages, the cost of this algorithm is given by

$$T_{\text{BKTREDUCE–SCATTER}}(p,n) = p\alpha_1 + \frac{p-1}{p}n(\beta_1 + \gamma)$$

6.4. Allreduce

Like the reduce–scatter the allreduce can be also be implemented using a BDE algorithm. This time at each step the entire vector is exchanged and added to the local result. This algorithm is given in Figure 13 and is illustrated in Figure 14. In the absence of network conflicts the cost of

[‡]Recall that short messages incur a latency of α_1.

DOI: 10.1002/cpe

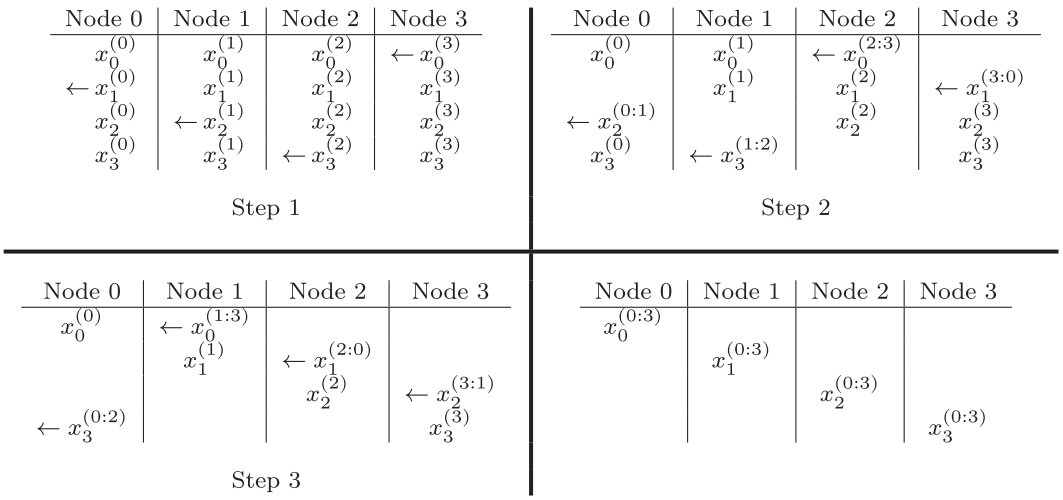


Figure 12. Bucket algorithm for reduce–scatter. Notation: $x_i^{(j_0:j_1)} = \sum_j x_i^{(j)}$ where $j_0 > j_1$ and $j \in \{j_0, j_0\% p + 1, \ldots, j_1\}$, and per cent denotes the integer modulus operation and p is the total number of nodes.

```
BDEALLREDUCE( x, left, right )

if left = right return
size = right − left + 1
mid = ⌊(left + right)/2⌋
if me ≤ mid
    partner = me + ⌊size/2⌋
else
    partner = me − ⌊size/2⌋

if me ≤ mid
    SEND( x, partner )
    RECV( tmp, partner )
    x = x + tmp
else
    SEND( x, partner )
    RECV( tmp, partner )
    x = x + tmp

if me ≤ mid
    BDEALLREDUCE( x, left, mid )
else
    BDEALLREDUCE( x, mid+1, right )
```

Figure 13. Bidirectional exchange algorithm for allreduce.

this algorithm is

$$T_{\text{BDEALLREDUCE}}(p, n) = \log(p)(\alpha_3 + n\beta_1 + n\gamma)$$

This cost attains the lower bound only for the latency component.

DOI: 10.1002/cpe

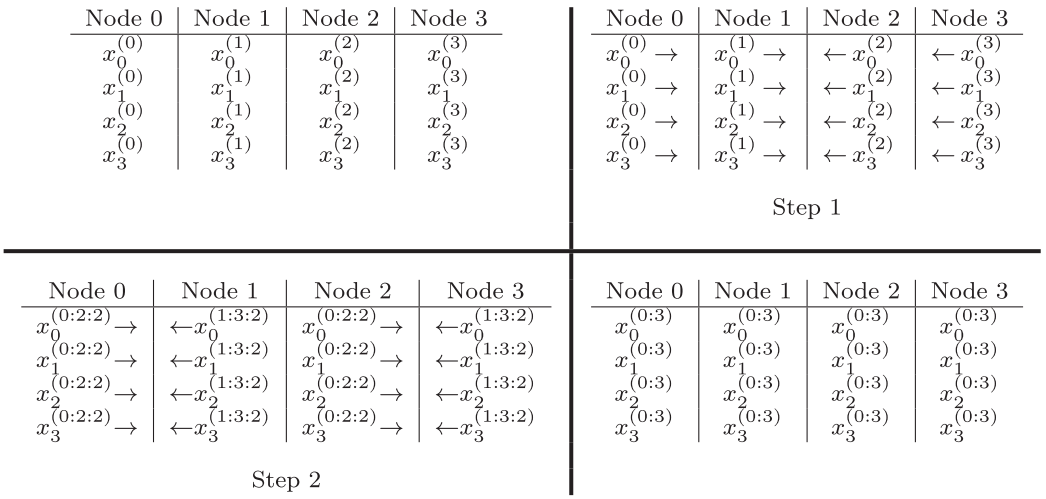


Figure 14. Bidirectional exchange algorithm for allreduce. In Step 1, bidirectional exchanges occur between the two pair of nodes 0 and 2, and 1 and 3.

7. MOVING ON

We now discuss how to pick and/or combine algorithms as a function of architecture, number of nodes, and vector length. We do so by presenting strategies for different types of architectures, building upon the algorithms that are already presented.

7.1. Linear arrays

On linear arrays, the MST BCAST and MST REDUCE algorithms achieve the lower bound for the α term while the BKT ALLGATHER and BKT REDUCE–SCATTER algorithms achieve the lower bound for the β term. The MST SCATTER and MST GATHER algorithms achieve the lower bounds for all vector lengths. BDE algorithms are undesirable since they require 2^d nodes and because they inherently incur network conflicts.

The following strategy provides simple algorithms that have merit in the extreme cases of short and long vector lengths. Figures 15 and 16 summarize this strategy where short and long vector algorithms can be used as 'building blocks' to compose different collective communication operations.

7.1.1. Broadcast

Short vectors: MST algorithm.
Long vectors: MST SCATTER followed by BKT ALLGATHER. The approximate cost is

$$T_{\text{SCATTER–ALLGATHER}}(p,n) = \lceil \log(p) \rceil \alpha_3 + \frac{p-1}{p} n\beta_1 + p\alpha_1 + \frac{p-1}{p} n\beta_1$$
$$= p\alpha_1 + \lceil \log(p) \rceil \alpha_3 + 2\frac{p-1}{p} n\beta_1$$

DOI: 10.1002/cpe

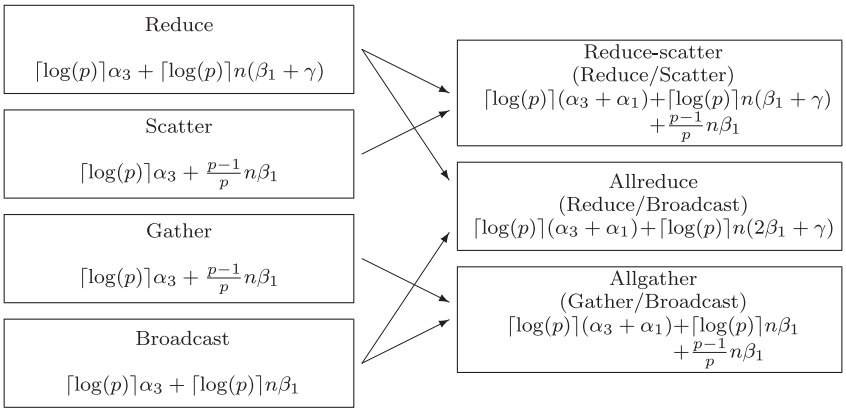


Figure 15. A building block approach to short vector algorithms on linear arrays.

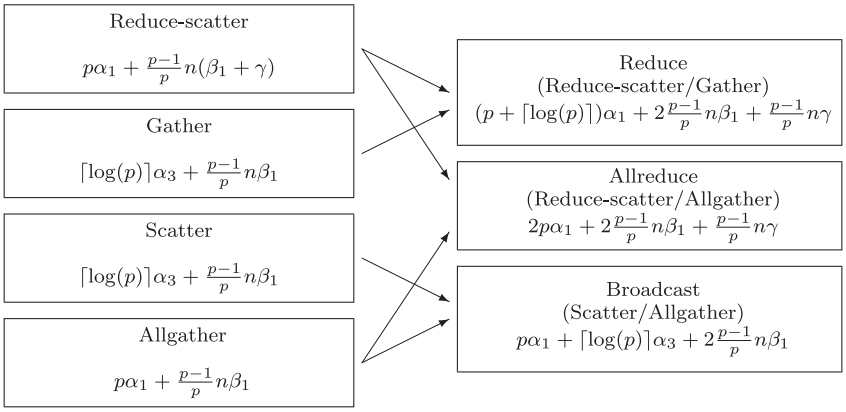


Figure 16. A building block approach to long vector algorithms on linear arrays.

As n gets large and the β term dominates, this cost is approximately $\lceil\log(p)\rceil/2$ times faster than the MST BCAST algorithm and within a factor of two of the lower bound.

7.1.2. Reduce

Short vectors: MST algorithm.
Long vectors: BKT REDUCE–SCATTER (the dual of the allgather) followed by a MST GATHER (the dual of the scatter). This time all the receives for the gather can be preposted before the reduce–scatter commences by observing that the completion of the reduce–scatter signals that all buffers for the gather are already available. Since the ready-receive sends can be used during the gather,

DOI: 10.1002/cpe

the cost becomes

$$T_{\text{REDUCE–SCATTER–GATHER}}(p,n) = p\alpha_1 + \frac{p-1}{p}n(\beta_1+\gamma) + \lceil\log(p)\rceil\alpha_1 + \frac{p-1}{p}n\beta_1$$
$$= (p + \lceil\log(p)\rceil)\alpha_1 + 2\frac{p-1}{p}n\beta_1 + \frac{p-1}{p}n\gamma$$

Again, the β term is within a factor of two of the lower bound while the γ term is optimal.

7.1.3. Scatter

Short vectors: MST algorithm.
Long vectors: Sending individual messages from the root to each of the other nodes. While the cost, $(p-1)\alpha_3 + ((p-1)/p)n\beta_1$, is clearly worse than the MST algorithm, in practice the β term has sometimes been observed to be smaller possibly because the cost of each message can be overlapped with those of other messages. We will call it the simple (SMPL) algorithm.

7.1.4. Gather

Same as scatter, in reverse.

7.1.5. Allgather

Short vectors: MST GATHER followed by MST BCAST. To reduce the α term, receives for the MST BCAST can be posted before the MST GATHER commences, yielding a cost of

$$T_{\text{GATHER–BCAST}}(p,n) = \lceil\log(p)\rceil\alpha_3 + \frac{p-1}{p}n\beta_1 + \lceil\log(p)\rceil\alpha_1 + \lceil\log(p)\rceil n\beta_1$$
$$\approx \lceil\log(p)\rceil(\alpha_1+\alpha_3) + (\lceil\log(p)\rceil+1)n\beta_1$$

This cost is close to the lower bound of $\lceil\log(p)\rceil\alpha$.
Long vectors: BKT algorithm.

7.1.6. Reduce–scatter

Short vectors: MST REDUCE followed by MST SCATTER. The receives for the MST SCATTER can be posted before the MST REDUCE commences, for a cost of

$$T_{\text{REDUCE–SCATTER}}(p,n) = \lceil\log(p)\rceil\alpha_3 + \lceil\log(p)\rceil n(\beta_1+\gamma) + \lceil\log(p)\rceil\alpha_1 + \frac{p-1}{p}n\beta_1$$
$$\approx \lceil\log(p)\rceil(\alpha_1+\alpha_3) + (\lceil\log(p)\rceil+1)n\beta_1 + \lceil\log(p)\rceil n\gamma$$

This cost is close to the lower bound of $\lceil\log(p)\rceil\alpha$.
Long vectors: BKT algorithm.

DOI: 10.1002/cpe

7.1.7. Allreduce

Short vectors: MST REDUCE followed by MST BCAST. The receives for the MST BCAST can be posted before the MST REDUCE commences, for a cost of

$$\begin{aligned} T_{\text{REDUCE–BCAST}}(p,n) &= \lceil \log(p) \rceil \alpha_3 + \lceil \log(p) \rceil n(\beta_1 + \gamma) + \lceil \log(p) \rceil \alpha_1 + \lceil \log(p) \rceil n\beta_1 \\ &= \lceil \log(p) \rceil (\alpha_1 + \alpha_3) + 2\lceil \log(p) \rceil n\beta_1 + \lceil \log(p) \rceil n\gamma \end{aligned}$$

This cost is close to the lower bound of $\lceil \log(p) \rceil \alpha$.

Long vectors: BKT REDUCE–SCATTER followed by BKT ALLGATHER. The approximate cost is

$$\begin{aligned} T_{\text{REDUCE–SCATTER–ALLGATHER}}(p,n) &= p\alpha_1 + \frac{p-1}{p} n(\beta_1 + \gamma) + p\alpha_1 + \frac{p-1}{p} n\beta_1 \\ &= 2p\alpha_1 + 2\frac{p-1}{p} n\beta_1 + \frac{p-1}{p} n\gamma \end{aligned}$$

This cost achieves the lower bound for the β and γ terms.

7.2. Multidimensional meshes

Next, we show that on multidimensional meshes the α term can be substantially improved for long vector algorithms, relative to linear arrays.

The key here is to observe that each row and column in a two-dimensional mesh forms a linear array and that all our collective communication operations can be formulated as performing the operation first within rows and then within columns, or vise versa.

For the two-dimensional mesh, we will assume that our p nodes physically form a $r \times c$ mesh and that the nodes are indexed in row-major order. For a mesh of dimension d, we will assume that the nodes are physically organized as a $d_0 \times d_1 \times \cdots \times d_{d-1}$ mesh.

7.2.1. Broadcast

Short vectors: MST algorithm within columns followed by MST algorithm within rows. The cost is

$$\begin{aligned} T_{\text{BCAST–BCAST}}(r,c,n) &= (\lceil \log(r) \rceil + \lceil \log(c) \rceil)(\alpha_3 + n\beta_1) \\ &\approx \lceil \log(p) \rceil (\alpha_3 + n\beta_1) \end{aligned}$$

Generalizing to a d-dimensional mesh yields an algorithm with a cost of

$$\sum_{k=0}^{d-1} \lceil \log(d_k) \rceil (\alpha_3 + n\beta_1) \approx \lceil \log(p) \rceil (\alpha_3 + n\beta_1)$$

Observe that the cost of the MST algorithm done successively in multiple dimensions yields approximately the same cost as performing it in just one dimension (e.g. a linear array).

DOI: 10.1002/cpe

Long vectors: MST SCATTER within columns, MST SCATTER within rows, BKT ALLGATHER within rows, BKT ALLGATHER within columns. The approximate cost is

$$\begin{aligned}T_{\text{SCATTER–SCATTER–ALLGATHER–ALLGATHER}}(r,c,n) &= \lceil\log(r)\rceil\alpha_3 + \frac{r-1}{r}n\beta_1 + \lceil\log(c)\rceil\alpha_3 + \frac{c-1}{c}\frac{n}{r}\beta_1 \\ &\quad + c\alpha_1 + \frac{c-1}{c}\frac{n}{r}\beta_1 + r\alpha_1 + \frac{r-1}{r}n\beta_1 \\ &= (c+r)\alpha_1 + (\lceil\log(c)\rceil + \lceil\log(r)\rceil)\alpha_3 + 2\frac{p-1}{p}n\beta_1 \\ &\approx (c+r)\alpha_1 + \lceil\log(p)\rceil\alpha_3 + 2\frac{p-1}{p}\beta_1\end{aligned}$$

As n gets large, and the β term dominates, this cost is still within a factor of two of the lower bound for β while the α term has been greatly reduced.

Generalizing to a d-dimensional mesh yields an algorithm with a cost of

$$\sum_{k=0}^{d-1} d_k\alpha_1 + \sum_{k=0}^{d-1}\lceil\log(d_k)\rceil\alpha_3 + 2\frac{p-1}{p}n\beta_1 \approx \sum_{k=0}^{d-1} d_k\alpha_1 + \lceil\log(p)\rceil\alpha_3 + 2\frac{p-1}{p}n\beta_1$$

7.2.2. Reduce

Short vectors: MST algorithm within rows followed by MST algorithm within columns. The cost is

$$\begin{aligned}T_{\text{REDUCE–REDUCE}}(r,c,n) &= (\lceil\log(c)\rceil + \lceil\log(r)\rceil)(\alpha_3 + n(\beta_1+\gamma)) \\ &\approx \lceil\log(p)\rceil(\alpha_3 + n(\beta_1+\gamma))\end{aligned}$$

Generalizing to a d-dimensional mesh yields an algorithm with a cost of

$$\sum_{k=0}^{d-1}\lceil\log(d_k)\rceil\alpha_3 + \sum_{k=0}^{d-1}\lceil\log(d_k)\rceil n(\beta_1+\gamma) \approx \lceil\log(p)\rceil(\alpha_3 + n(\beta_1+\gamma))$$

Long vectors: BKT REDUCE–SCATTER within rows, BKT REDUCE–SCATTER within columns, MST GATHER within columns, MST GATHER within rows. The approximate cost is

$$\begin{aligned}&T_{\text{REDUCE–SCATTER–REDUCE–SCATTER–GATHER–GATHER}}(r,c,n) \\ &\quad= c\alpha_1 + \frac{c-1}{c}n\beta_1 + \frac{c-1}{c}n\gamma + r\alpha_1 + \frac{r-1}{r}\frac{n}{c}\beta_1 + \frac{r-1}{r}\frac{n}{c}\gamma \\ &\qquad + \lceil\log(r)\rceil\alpha_3 + \frac{r-1}{r}\frac{n}{c}\beta_1 + \lceil\log(c)\rceil\alpha_3 + \frac{c-1}{c}n\beta_1 \\ &\quad= (c+r)\alpha_1 + (\lceil\log(r)\rceil + \lceil\log(c)\rceil)\alpha_3 + 2\frac{p-1}{p}n\beta_1 + \frac{p-1}{p}n\gamma \\ &\quad\approx (c+r)\alpha_1 + \lceil\log(p)\rceil\alpha_3 + 2\frac{p-1}{p}n\beta_1 + \frac{p-1}{p}n\gamma\end{aligned}$$

DOI: 10.1002/cpe

Generalizing to a d-dimensional mesh yields an algorithm with a cost of

$$\sum_{k=0}^{d-1} d_k\alpha_1 + \sum_{k=0}^{d-1} \lceil\log(d_k)\rceil\alpha_3 + 2\frac{p-1}{p}n\beta_1 + \frac{p-1}{p}n\gamma$$
$$\approx \sum_{k=0}^{d-1} d_k\alpha_1 + \lceil\log(p)\rceil\alpha_3 + 2\frac{p-1}{p}n\beta_1 + \frac{p-1}{p}n\gamma$$

As n gets large, and the β term dominates, this cost is within a factor of two of the lower bound for β. The γ term is optimal.

7.2.3. Scatter

Short vectors: MST algorithms successively in each of the dimensions.
Long vectors: SMPL algorithms successively in each of the dimensions.

7.2.4. Gather

Same as scatter, in reverse.

7.2.5. Allgather

Short vectors: MST GATHER within rows, MST BCAST within rows, MST GATHER within columns, MST BCAST within columns. Again, preposing can be used to reduce the α term for the broadcasts, for a total cost of

$$\begin{aligned} T_{\text{GATHER–BCAST–GATHER–BCAST}}(r,c,n) &= \lceil\log(c)\rceil\alpha_3 + \frac{c-1}{c}\frac{n}{r}\beta_1 + \lceil\log(c)\rceil\alpha_1 + \lceil\log(c)\rceil\frac{n}{r}\beta_1 \\ &\quad + \lceil\log(r)\rceil\alpha_3 + \frac{r-1}{r}n\beta_1 + \lceil\log(r)\rceil\alpha_1 + \log(r)n\beta_1 \\ &= (\lceil\log(r)\rceil + \lceil\log(c)\rceil)(\alpha_1+\alpha_3) \\ &\quad + \left(\frac{p-1}{p} + \frac{\lceil\log(c)\rceil}{r} + \lceil\log(r)\rceil\right)n\beta_1 \\ &\approx \lceil\log(p)\rceil(\alpha_1+\alpha_3) + \left(\frac{p-1}{p} + \frac{\lceil\log(c)\rceil}{r} + \lceil\log(r)\rceil\right)n\beta_1 \end{aligned}$$

Generalizing to a d-dimensional mesh yields an algorithm with a cost of

$$\sum_{k=0}^{d-1} \lceil\log(d_k)\rceil(\alpha_1+\alpha_3) + \left(\frac{p-1}{p} + \frac{\lceil\log(d_{d-1})\rceil}{d_0d_1\cdots d_{d-2}} + \cdots + \frac{\lceil\log(d_1)\rceil}{d_0} + \lceil\log(d_0)\rceil\right)n\beta_1$$
$$\approx \lceil\log(p)\rceil(\alpha_1+\alpha_3) + \left(\frac{p-1}{p} + \frac{\lceil\log(d_{d-1})\rceil}{d_0d_1\cdots d_{d-2}} + \cdots + \frac{\lceil\log(d_1)\rceil}{d_0} + \lceil\log(d_0)\rceil\right)n\beta_1$$

DOI: 10.1002/cpe

Note that the α term remains close to the lower bound of $\lceil \log(p) \rceil \alpha$ while the β term has been reduced!

Long vectors: BKT ALLGATHER within rows, BKT ALLGATHER within columns. The cost is

$$T_{\text{ALLGATHER–ALLGATHER}}(r, c, n) = c\alpha_1 + \frac{c-1}{c}\frac{n}{r}\beta_1 + r\alpha_1 + \frac{r-1}{r}n\beta_1$$
$$= (c + r)\alpha_1 + \frac{p-1}{p}n\beta_1$$

Generalizing to a d-dimensional mesh yields an algorithm with a cost of

$$\sum_{k=0}^{d-1} d_k\alpha_1 + \frac{p-1}{p}n\beta_1.$$

7.2.6. *Reduce–scatter*

Short vectors: MST REDUCE within columns, MST SCATTER within columns, MST REDUCE within rows, MST SCATTER within rows. Preposing the receives for the scatter operations yields a total cost of

$$\begin{aligned}
&T_{\text{REDUCE–SCATTER–REDUCE–SCATTER}}(r, c, n) \\
&= \lceil \log(r) \rceil \alpha_3 + \lceil \log(r) \rceil n(\beta_1 + \gamma) + \lceil \log(r) \rceil \alpha_1 \\
&\quad + \frac{r-1}{r}\frac{n}{c}\beta_1 + \lceil \log(c) \rceil \alpha_3 + \frac{c-1}{c}n(\beta_1 + \gamma) + \lceil \log(c) \rceil \alpha_1 + \log(c)n\beta_1 \\
&= (\lceil \log(r) \rceil + \lceil \log(c) \rceil)(\alpha_1 + \alpha_3) + \left(\frac{p-1}{p} + \frac{\lceil \log(c) \rceil}{r} + \lceil \log(r) \rceil \right) n\beta_1 \\
&\quad + \left(\frac{\lceil \log(c) \rceil}{r} + \lceil \log(r) \rceil \right) n\gamma \\
&\approx \lceil \log(p) \rceil (\alpha_1 + \alpha_3) + \left(\frac{p-1}{p} + \frac{\lceil \log(c) \rceil}{r} + \lceil \log(r) \rceil \right) n\beta_1 + \left(\frac{\lceil \log(c) \rceil}{r} + \lceil \log(r) \rceil \right) n\gamma
\end{aligned}$$

Generalizing to a d-dimensional mesh yields an algorithm with a cost of

$$\begin{aligned}
&\sum_{k=0}^{d-1} \lceil \log(d_k) \rceil (\alpha_1 + \alpha_3) + \left(+ \frac{\lceil \log(d_0) \rceil}{d_1 \cdots d_{d-1}} + \cdots + \frac{\lceil \log(d_{d-2}) \rceil}{d_{d-1}} + \lceil \log(d_{d-1}) \rceil \right) n\beta_1 \\
&\quad + \left(\frac{\lceil \log(d_0) \rceil}{d_1 \cdots d_{d-1}} + \cdots + \frac{\lceil \log(d_{d-2}) \rceil}{d_{d-1}} + \lceil \log(d_{d-1}) \rceil \right) n\gamma \\
&\approx \lceil \log(p) \rceil (\alpha_1 + \alpha_3) + \left(+ \frac{\lceil \log(d_0) \rceil}{d_1 \cdots d_{d-1}} + \cdots + \frac{\lceil \log(d_{d-2}) \rceil}{d_{d-1}} + \lceil \log(d_{d-1}) \rceil \right) n\beta_1 \\
&\quad + \left(\frac{\lceil \log(d_0) \rceil}{d_1 \cdots d_{d-1}} + \cdots + \frac{\lceil \log(d_{d-2}) \rceil}{d_{d-1}} + \lceil \log(d_{d-1}) \rceil \right) n\gamma
\end{aligned}$$

DOI: 10.1002/cpe

Note that the α term remains close to the lower bound of $\lceil \log(p) \rceil \alpha$ while both the β and γ terms have been reduced!

Long vectors: BKT REDUCE–SCATTER within rows, BKT REDUCE–SCATTER within columns. The cost is

$$T_{\text{REDUCE–SCATTER–REDUCE–SCATTER}}(r, c, n) = r\alpha_1 + \frac{r-1}{r}\frac{n}{c}(\beta_1 + \gamma) + c\alpha_1 + \frac{c-1}{c}n(\beta_1 + \gamma)$$
$$= (r + c)\alpha_1 + \frac{p-1}{p}n(\beta_1 + \gamma)$$

Generalizing to a d-dimensional mesh yields an algorithm with a cost of

$$\sum_{k=0}^{d-1} d_k \alpha_1 + \frac{p-1}{p}n(\beta_1 + \gamma)$$

7.2.7. *Allreduce*

Short vectors: MST REDUCE followed by MST BCAST (both discussed above), preposting the receives for the broadcast. The approximate cost is

$$T_{\text{REDUCE–BCAST}}(d, n) = 2\sum_{k=0}^{d-1} \lceil \log(d_k) \rceil (\alpha_1 + \alpha_3)$$
$$+ 2\left(\frac{\lceil \log(d_0) \rceil}{d_1 \cdots d_{d-1}} + \cdots + \frac{\lceil \log(d_{d-2}) \rceil}{d_{d-1}} + \lceil \log(d_{d-1}) \rceil\right) n\beta_1$$
$$+ \left(\frac{\lceil \log(d_0) \rceil}{d_1 \cdots d_{d-1}} + \cdots + \frac{\lceil \log(d_{d-2}) \rceil}{d_{d-1}} + \lceil \log(d_{d-1}) \rceil\right) n\gamma$$

Long vectors: BKT REDUCE–SCATTER followed by BKT ALLGATHER (both discussed above). The approximate cost is

$$T_{\text{REDUCE–SCATTER–ALLGATHER}}(d, n) = 2\sum_{k=0}^{d-1} d_k \alpha_1 + 2\frac{p-1}{p}n\beta_1 + \frac{p-1}{p}n\gamma$$

This cost achieves the lower bound for the β and γ terms.

7.3. Hypercubes

We now argue that the discussion on algorithms for multidimensional meshes includes all algorithms already discussed for hypercubes. Recall that a d-dimensional hypercube is simply a d-dimensional mesh with $d_0 = \cdots = d_{d-1} = 2$. Now, the short vector algorithm for broadcasting successively in each of the dimensions becomes the MST BCAST on hypercubes. The long vector algorithm for allgather that successively executes a BKT algorithm in each dimension is equivalent to a BDE

DOI: 10.1002/cpe

algorithm on hypercubes. Similar connections can be made for other algorithms discussed for multidimensional meshes.

The conclusion is that we can concentrate on optimizing algorithms for multidimensional meshes. A by-product of the analyses will be optimized algorithms for hypercubes.

7.4. Fully connected architectures

Fully connected architectures can be viewed as multidimensional meshes so that, as noted for hypercube architectures, it suffices to analyze the optimization of algorithms for multidimensional meshes.

8. STRATEGIES FOR ALL VECTOR LENGTHS

We have developed numerous algorithms for the short and long vector cases. They have been shown to be part of a consistent family rather than a bag full of algorithms. The natural next question becomes how to deal with intermediate length vectors. A naive solution would be to determine the crossover point between the short and long vector costs and switch algorithms at that crossover point. In this section, we show that one can do much better with 'hybrid' algorithms. Key to this approach is the recognition that all collective communications have in common the property that the operation performed among all nodes yields the same result as when the nodes are logically viewed as a two-dimensional mesh, and the operation is performed first in one dimension and next in the second dimension. This observation leaves the possibility of using a different algorithm in each of the two dimensions.

8.1. A prototypical example: broadcast

Consider a broadcast on a $r \times c$ mesh of nodes. A broadcast among all nodes can then be implemented as

Step 1: A broadcast within the row of nodes that includes the original root.
Step 2: Simultaneous broadcasts within columns of nodes where the roots of the nodes are in the same row as the original root.

For each of these two steps, a different broadcast algorithm can be chosen.

Now, consider the case where in Step 1 a long vector algorithm is chosen: MST SCATTER followed by BKT ALLGATHER. It is beneficial to orchestrate the broadcast as

Step 1a: MST SCATTER within the rows of nodes that includes the original root.
Step 2: Simultaneous broadcasts within columns of nodes where the roots of the nodes are in the same row as the original root.
Step 1b: Simultaneous BKT ALLGATHER within rows of nodes.

The benefit is that now in Step 2 the broadcasts involve vectors of length approximately n/c. This algorithm is illustrated in Figure 17.

DOI: 10.1002/cpe

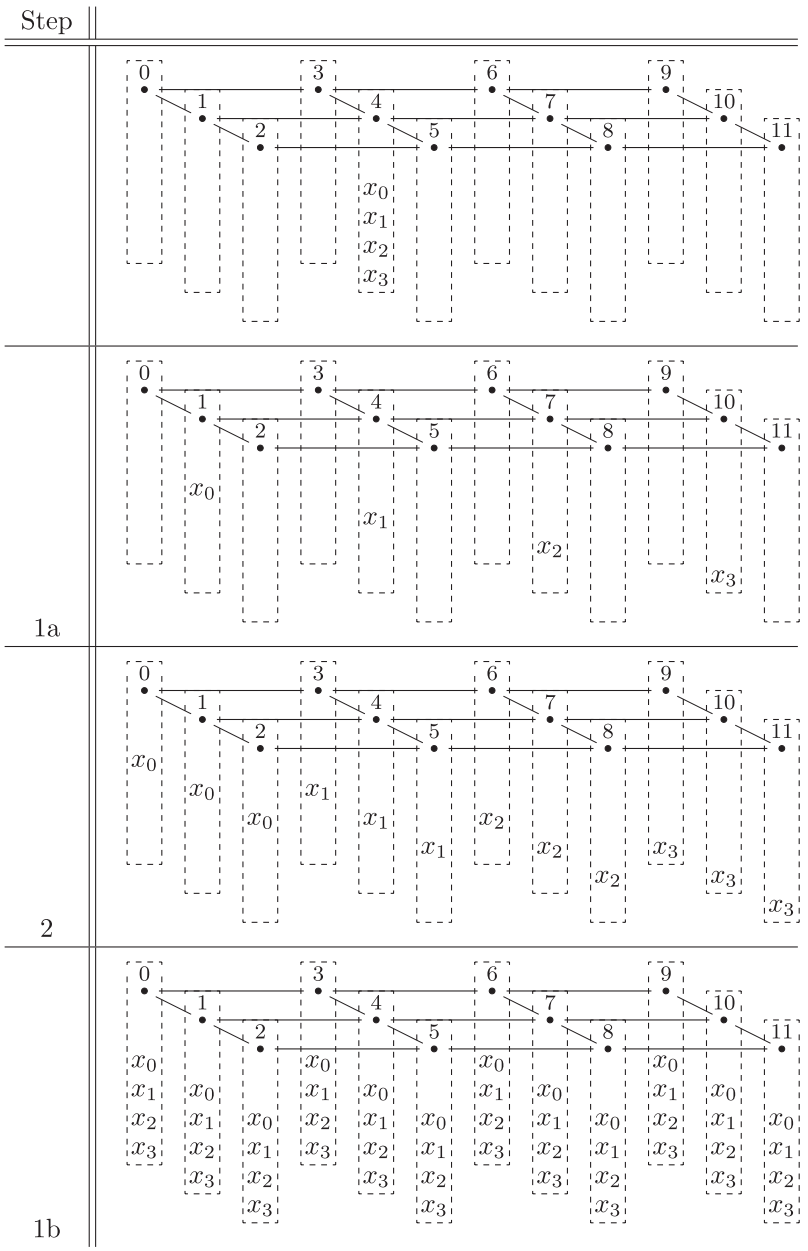


Figure 17. Broadcast on a (logical) two-dimensional mesh implemented by (1a) a scatter within the row that includes the root, followed by (2) a broadcast within the columns, followed by (1b) an allgather within rows.

DOI: 10.1002/cpe

Option	Algorithm	Cost
1	MST BCAST (all nodes).	$\lceil\log(p)\rceil\alpha_3 + \lceil\log(p)\rceil n\beta_1$
2	Step 1: MST BCAST, Step 2: MST BCAST.	$(\lceil\log(r)\rceil + \lceil\log(c)\rceil)\alpha_3$ $+ (\lceil\log(r)\rceil + \lceil\log(c)\rceil)n\beta_1$
3	Step 1a: MST SCATTER, Step 2: MST BCAST, Step 1b: BKT ALLGATHER.	$c\alpha_1 + (\lceil\log(r)\rceil + \lceil\log(c)\rceil)\alpha_3$ $+ (\frac{\lceil\log(r)\rceil}{c} + 2\frac{c-1}{c})n\beta_1$
4	Step 1: MST BCAST, Step 2: MST SCATTER–BKT ALLGATHER.	$c\alpha_1 + (\lceil\log(r)\rceil + \lceil\log(c)\rceil)\alpha_3$ $+ (\lceil\log(r)\rceil + 2\frac{c-1}{c})n\beta_1$
5	Step 1a: MST SCATTER, Step 2: MST SCATTER–BKT ALLGATHER, Step 1b: BKT ALLGATHER.	$(r+c)\alpha_1 + (\lceil\log(r)\rceil + \lceil\log(c)\rceil)\alpha_3$ $+ 2\frac{p-1}{p}n\beta_1$ (assuming $r \neq 1$ and $c \neq 1$)
6	Step 1: MST SCATTER–BKT ALLGATHER, Step 2: MST SCATTER–BKT ALLGATHER.	$(r+c)\alpha_1 + (\lceil\log(r)\rceil + \lceil\log(c)\rceil)\alpha_3$ $+ 2(\frac{r-1}{r} + \frac{c-1}{c})n\beta_1$
7	MST SCATTER–BTK ALLGATHER (all nodes).	$p\alpha_1 + \lceil\log(p)\rceil\alpha_3 + 2\frac{p-1}{p}n\beta_1$

Figure 18. Various approaches to implementing a broadcast on an $r \times c$ mesh. The cost analysis assumes no network conflicts occur.

For fixed r and c, combinations of short and long vector algorithms for Steps 1 and 2 are examined in Figure 18. Note that:

- Option 2 is never better than Option 1 since $\lceil\log(r)\rceil + \lceil\log(c)\rceil \geqslant \lceil\log(p)\rceil$.
- Option 4 is never better than Option 3. The observation here is that if a SCATTER–ALLGATHER broadcast is to be used, it should be as part of Step 1a–1b so that the length of the vectors to be broadcast during Step 2 is reduced.
- Option 5 is generally better than Option 6 since the α terms are identical while the β term is smaller for Option 5 than for Option 6 (since $n/p \leqslant n/r$).
- Option 5 is generally better than Option 7 because the β terms are identical while the α term has been reduced since $r + c \leqslant p$ (when p is not prime).

Thus, we find that there are three algorithms of interest:

- *Option 1*: MST BCAST among all nodes. This algorithm is best when the vector lengths are short.
- *Option 3:* MST SCATTER–MST BCAST–BKT ALLGATHER. This algorithm is best when the vector length is such that the scatter leaves subvectors that are considered to be small when broadcast among r nodes.
- *Option 5*: MST SCATTER–MST SCATTER–BKT ALLGATHER–BKT ALLGATHER. This algorithm is best when the vector length is such that the scatter leaves subvectors that are considered to be long when broadcast among r nodes if multiple integer factorizations of p exist.

In Figure 19(a), we show the predicted cost, in time, of each of these options on a fully connected machine with $p = 256$, $r = c = 16$, $\alpha_1 \approx 2 \times 10^{-6}$, $\beta_1 \approx 1 \times 10^{-9}$, and $\gamma \approx 1 \times 10^{-10}$. The graph clearly shows how the different options trade lowering the α term for increasing the β term, the extremes being the MST SCATTER–BKT ALLGATHER, with the greatest α term and lowest β term, and the MST BCAST, with the lowest α term and greatest β term.

DOI: 10.1002/cpe

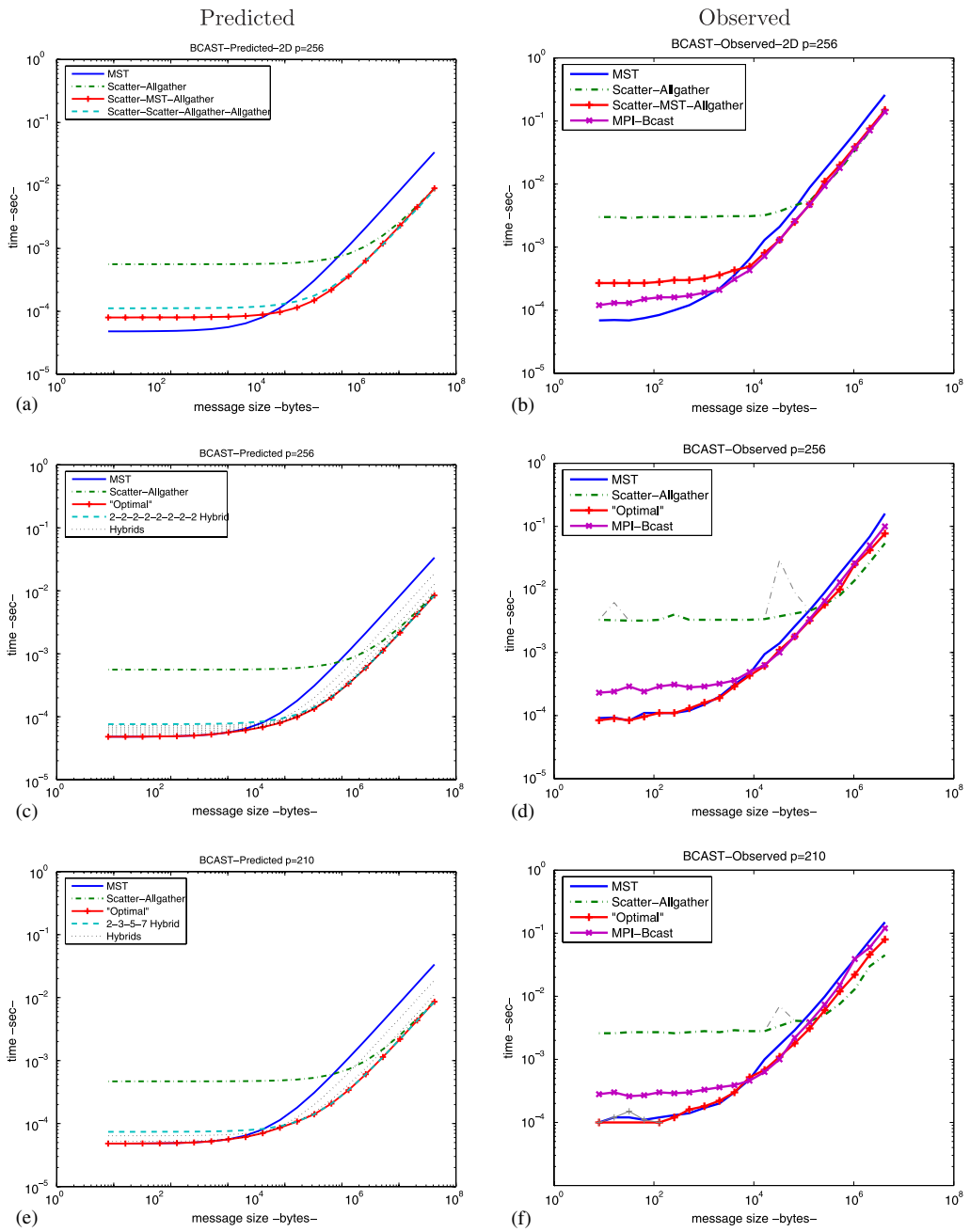


Figure 19. Comparing experimental results with predicted results.

DOI: 10.1002/cpe

Naturally, many combinations of r and c can be chosen, and the technique can be extended to more than two dimensions, which we discuss next.

8.2. Optimal hybridization on hypercubes

We now discuss how to optimally combine short and long vector algorithms for the broadcast operation. Our strategy is to develop theoretical results for hypercubes, first reported in [7] for the allgather operation. This theoretical result will then motivate heuristics for multidimensional meshes discussed in Section 8.3.

Assume that $p = 2^d$ and that the nodes form a hypercube architecture (or, alternatively, are fully connected). We assume that a vector of length n is to be broadcast and, for simplicity, that n is an integer multiple of p.

A *strategy* will be indicated by an integer D, $0 \leqslant D < d$, and two vectors, $(a_0, a_1, \ldots, a_{D-1})$ and $(d_0, d_1, \ldots, d_{D-1})$. The idea is that the processors are viewed as a D-dimensional mesh, that the ith dimension of that mesh has d_i nodes that themselves form a hypercube, and that $a_i \in \{\text{long, short}\}$ indicates whether a long or short vector algorithm is to be used in the ith dimension.

- If $a_i = \text{long}$, then a long-vector algorithm is used in the ith dimension of the mesh. The vector is scattered among d_i processors, and each piece (now reduced in length by a factor d_i) is broadcast among the remaining dimensions of the mesh using the strategy $(a_{i+1}, \ldots, a_{D-1})$, $(d_{i+1}, \ldots, d_{D-1})$, after which the result is collected *via* BKT ALLGATHER.
- If $a_i = \text{short}$, then MST BCAST is used in the ith dimension of the mesh, and a broadcast among the remaining dimensions of the mesh using the strategy $(a_{i+1}, \ldots, a_{D-1})$, $(d_{i+1}, \ldots, d_{D-1})$ is employed.

The cost given a strategy is given by the inductively defined cost function

$$C(n, (a_0, \ldots, a_{D-1}), (d_0, \ldots, d_{D-1}))$$
$$= \begin{cases} 0 & \text{if } D = 0 \\ d_0\alpha_1 + \log(d_0)\alpha_3 + 2\dfrac{d_0 - 1}{d_0} n\beta_1 & \\ \quad + C\left(\dfrac{n}{d_0}, (a_1, \ldots, a_{D-1}), (d_1, \ldots, d_{D-1})\right) & \text{if } D > 0 \text{ and } a_0 = \text{long} \\ \log(d_0)\alpha_3 + \log(d_0)n\beta_1 & \\ \quad + C(n, (a_1, \ldots, a_{D-1}), (d_1, \ldots, d_{D-1})) & \text{if } D > 0 \text{ and } a_0 = \text{short} \end{cases}$$

Some simple observations are:

- Assume $d_i > 2$ (but a power of two). Then,

$$C(n, (a_0, \ldots, a_{i-1}, \underbrace{a_i,}\ a_{i+1}, \ldots, a_{D-1}), (d_0, \ldots, d_{i-1}, \underbrace{d_i,}\ d_{i+1}, \ldots, d_{D-1}))$$
$$\geqslant$$
$$C(n, (a_0, \ldots, a_{i-1}, \overbrace{a_i, a_i}, a_{i+1}, \ldots, a_{D-1}), (d_0, \ldots, d_{i-1}, \overbrace{2, \frac{d_i}{2}}, d_{i+1}, \ldots, d_{D-1}))$$

DOI: 10.1002/cpe

This observation tells us that an optimal (minimal cost) strategy can satisfy the restriction that $D = d$ and $d_0 = \cdots = d_{d-1} = 2$.

- Assume $a_i =$ short and $a_{i+1} =$ long. Then,

$$C(n, (a_0, \ldots, a_{i-1}, \underbrace{a_i, a_{i+1},} a_{i+2}, \ldots, a_{D-1}), (d_0, \ldots, d_{i-1}, \underbrace{d_i, d_{i+1},} d_{i+2}, \ldots, d_{D-1}))$$

$$\geqslant$$

$$C(n, (a_0, \ldots, a_{i-1}, \overbrace{a_{i+1}, a_i,} a_{i+2}, \ldots, a_{D-1}), (d_0, \ldots, d_{i-1}, \overbrace{d_{i+1}, d_i,} d_{i+2}, \ldots, d_{D-1}))$$

This observation tells us that an optimal strategy can satisfy the restriction that a short-vector algorithm is never used before a long-vector algorithm.

Together these two observations indicate that an optimal strategy exists among the strategies that view the nodes as a d-dimensional mesh, with two nodes in each dimension, and have the form $(a_0, \ldots, a_{k-1}, a_k, \ldots, a_{d-1})$ where $a_i =$ long if $i < k$ and $a_i =$ short if $i \geqslant k$.

An optimal strategy can thus be determined by choosing k_{opt}, the number of times a long vector algorithm is chosen. The cost of such a strategy is now given by

$$\begin{aligned} C(n,k) &= 2^k\alpha_1 + \log(2^k)\alpha_3 + 2\frac{2^k-1}{2^k}n\beta_1 + \log(2^{d-k})\alpha_3 + \log(2^{d-k})\frac{n}{2^k}\beta_1 \\ &= 2^k\alpha_1 + k\alpha_3 + 2\frac{2^k-1}{2^k}n\beta_1 + (d-k)\alpha_3 + (d-k)\frac{n}{2^k}\beta_1 \\ &= 2^k\alpha_1 + d\alpha_3 + (2^{k+1} - 2 + d - k)\frac{n}{2^k}\beta_1 \end{aligned}$$

Let us now examine $C(n,k)$ vs $C(n,k+1)$:

$$\begin{aligned} C(n,k+1) - C(n,k) &= \left(2^{k+1}\alpha_1 + d\alpha_3 + (2^{k+2} - 2 + d - k - 1)\frac{n}{2^{k+1}}\beta_1\right) \\ &\quad - \left(2^k\alpha_1 + d\alpha_3 + (2^{k+1} - 2 + d - k)\frac{n}{2^k}\beta_1\right) \\ &= 2^k\alpha_1 + (1 - d + k)\frac{n}{2^{k+1}}\beta_1 \end{aligned} \tag{1}$$

Next, we will show that if $C(n,k) \leqslant C(n,k+1)$ then $C(n,k+1) \leqslant C(n,k+2)$:

$$\begin{aligned} C(n,k+2) - C(n,k+1) &= 2^{k+1}\alpha_1 + (2 - d + k)\frac{n}{2^{k+2}}\beta_1 \\ &= 2(2^k\alpha_1) + \frac{2-d+k}{2}\frac{n}{2^{k+1}}\beta_1 \geqslant 2^k\alpha_1 + (2-d+k)\frac{n}{2^{k+1}}\beta_1 \\ &\geqslant 2^k\alpha_1 + (1-d+k)\frac{n}{2^{k+1}}\beta_1 = C(n,k+1) - C(n,k) \end{aligned}$$

This result shows that $C(n,k)$ as a function of k is concave up. Thus, k_{opt} can be chosen to equal the smallest non-negative integer k such that $C(n,k) \leqslant C(n,k+1)$, which is equivalent to the smallest non-negative integer k for which the expression in (1) becomes non-negative.

DOI: 10.1002/cpe

The above analysis supports the following simple observations:

- In each of the $\log(p)$ dimensions, it must be decided whether to use the long or short vector algorithm.
- It makes sense to use the long vector algorithm first since it reduces the vector length for subsequent dimensions.
- The condition discussed above indicates when the short vector algorithm should be used for all subsequent dimensions.

Since hybrid algorithms can be composed in much the same way for all the discussed collective communications, a similar result can be established for the hybridization of all collective communication algorithms on hypercubes.

In Figure 19(c) we show the predicted cost, in time, of various strategies when the same parameters were used as the graph on its left. In the graph '2-2-2-2-2-2-2-2 Hybrid' indicates the strategy that uses the long vector algorithm in all dimensions and 'Optimal' indicates the optimal strategy discussed above. Various other hybrids, which represent different choices for k in the above discussion, are also plotted.

8.3. A strategy for designing tunable libraries

Developing an optimal strategy, supported by theory, for multidimensional mesh architectures and fully connected architectures with non-power-of-two numbers of nodes is at the very least non-trivial and possibly intractable. As a result, we advocate heuristics that are guided by the theory that were developed for the ideal hypercube architecture in the previous section.

Assuming an architecture with p nodes. The basic hybrid algorithm for the broadcast, motivated by Section 8.2, is given with:

- Choose $d_0 \times d_1 \times \cdots \times d_{D-1}$, an integer factorization of p.
- Choose k, $0 \leqslant k < D$.
- *Step 1a*: MST SCATTER within the first k dimensions.
- *Step 2*: MST BCAST within the remaining $D - k$ dimensions.
- *Step 1b*: BKT ALLGATHER within the first k dimensions, in opposite order.

The parameters that need to be chosen, based on the architecture parameters α and β as well as the vector length n, are the integer factorization, k, and the order in which dimensions are picked. Depending on the architecture, it may also be necessary to factor in network conflicts. For example, one could view a linear array with p nodes as a $r \times c$ mesh, but then in one of the dimensions network conflicts would occur.

There are many variants to this theme. For example, one could restrict oneself to three integer factorizations: $1 \times p, r \times c$, and $p \times 1$, for $r \approx c \approx \sqrt{p}$ and only consider the three options mentioned in Section 8.1. One could also carefully model all options and pick a strategy based on the minimal predicted cost. What is important is that our exposition leading up to this section creates a naturally parameterized family of options.

In Figure 19(e), we show the predicted cost, in time, of various hybrids on a mesh with $p = 2 \times 3 \times 5 \times 7 = 256$. In that graph '2-3-5-7 Hybrid' indicates the hybrid that executes the long vector algorithm in each dimension while 'Optimal' indicates an exhaustive search through all

DOI: 10.1002/cpe

strategies. What the graph demonstrates is that on higher dimensional meshes, or fully connected architectures that are viewed as higher dimensional meshes, performance similar to that observed on hypercubes can be attained by picking a reasonable strategy along the lines that we outlined above.

All other operations can be similarly treated since the general principle behind all algorithms is the same: hybrids are created by nesting long vector and short vector algorithms.

9. EXPERIMENTS

We now demonstrate briefly that the techniques that have been discussed so far have merit in practice. Further evidence, from the Intel Touchstone Delta and Paragon systems, can be found in [20–22]. Rather than exhaustively showing performance for all operations, we focus on the broadcast and reduce–scatter operations.

9.1. Testbed architecture

The architecture on which experiments were conducted in Figures 19 and 20 is a Cray-Dell PowerEdge Linux Cluster operated by the Texas Advanced Computing Center (TACC). At the time the experiments were conducted, this cluster contained 768 3.06 and 256 3.2 GHz Xeon/Pentium 4 processors within 512 Dell dual-processor PowerEdge 1750 compute nodes where each compute node had 2 GB of memory. A Myrinet-2000 switch fabric, employing PCI-X interfaces, interconnected the nodes with a sustainable point-to-point bandwidth of 250 MB/s. The experiments were conducted with the MPICH-GM library 1.2.5 . . . 12, GM 2.1.2, and Intel compiler 7.0 running Red Hat Linux 7.1.

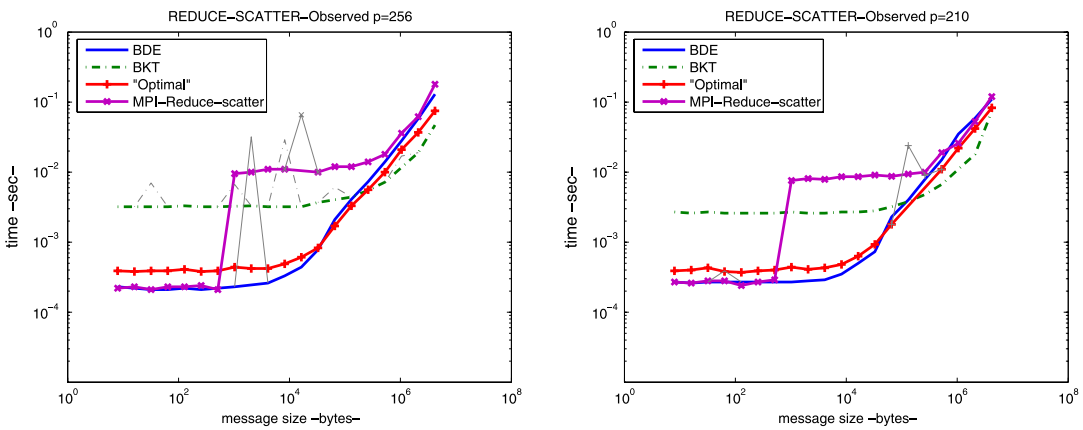


Figure 20. Observed performance of various hybrid reduce–scatter algorithms. Left: Performance of hybrids on 256 nodes viewed as a hypercube or fully connected architecture. Right: Performance on 210 nodes viewed as a $2 \times 3 \times 5 \times 7$ mesh.

DOI: 10.1002/cpe

9.2. Implementation

The algorithms discussed in this paper were implemented using MPI point-to-point send and receive calls. They are available as part of the InterCol library developed at UT-Austin which is available at `http://www.tacc.utexas.edu/resources/tools/intercol.php`

9.3. Results

In Figure 19, we show performance attained by our broadcast implementations. When the data were collected, it was somewhat noisy. This noise is indicated in the graphs by the 'thin' lines and was removed from the 'thick' lines so that the predicted and observed data could be more easily compared. We note that the machine on which the data were collected was not a dedicated machine, which explains the noise. Qualitatively the predicted and observed data match quite closely. It is interesting to note that the pure SCATTER–ALLGATHER algorithm performs better than predicted relative to the hybrids. This result can be attributed to the fact that the architecture is not a truly fully connected architecture which means that network conflicts occurred as logical higher dimensional meshes were mapped to the physical architecture. Clearly, a truly optimal hybrid algorithm would switch to the SCATTER–ALLGATHER algorithm at some point.

In Figure 20, we show the performance attained by our reduce–scatter implementations. The story is quite similar to that reported for the broadcast. Most interesting is the performance curve for the MPICH implementation. That library appears to create a hybrid from two algorithms: BDE and BKT. However, their implementation of BKT REDUCE–SCATTER appears to be less efficient and clearly the crossover point was not optimized correctly. We note that such libraries are continuously improved and that the reported performance may not be indicative of the current implementation. Despite such continual updates of libraries, nearly all implementations use a combination of algorithms presented in this paper with varying crossover points.

In these figures, the 'Optimal' performance curve was obtained by applying the heuristic described in Section 8.3 with the estimates of α and β that were used for the predicted data in Figure 19.

The performance data that we have reported is representative of data we observed for the other collective communication operations.

10. CONCLUSION

Many of the techniques described in this paper date back to the InterCom project at UT-Austin in the early 1990s. That project produced a collective communication library specifically for the Intel Touchstone Delta and Paragon parallel supercomputers [28], and it was used by MPI implementations on those platforms. This paper shows that those early algorithms still represent the state of the art.

The discussion in Section 8.3 is a key contribution of this paper. It provides a framework to those who pursue automatic optimization of collective communication libraries by providing a parameterized family of algorithms rather than an *ad hoc* collection.

Clearly, the model that we use to analyze and describe the algorithms is restricting. For example, architectures that can send messages simultaneously in multiple directions exist such as the IBM

DOI: 10.1002/cpe

Blue Gene/L [29]. It is our belief that the descriptions given in this paper can be easily modified to take advantage of such architectural features [30].

Collective communication is not just an issue for distributed-memory architectures that are commonly used for large scientific applications. As multi-core technology evolves to the point where there will be many cores on a chip, it is likely that each core will have their own local memory, and collective communication will be used to reduce memory contention. Thus, the simple algorithms discussed in this paper may find new uses.

ACKNOWLEDGEMENTS

Some of the material in this paper was taken from an unpublished manuscript by Payne *et al.* [31]. We gratefully acknowledge the contributions of these researchers to the original InterCom project and that manuscript. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation (NSF).

REFERENCES

1. Fox G, Johnson M, Lyzenga G, Otto S, Salmon J, Walker D. *Solving Problems on Concurrent Processors*. vol. I. Prentice-Hall: Englewood Cliffs, NJ, 1988.
2. Ho C-T, Johnsson SL. Distributed routing algorithms for broadcasting and personalized communication in hypercubes. *Proceedings of the 1986 International Conference on Parallel Processing*. IEEE: New York, 1986; 640–648.
3. Johnsson SL. Communication efficient basic linear algebra computations on hypercube architectures. *Journal of Parallel and Distributed Computing* 1987; **4**:133–172.
4. Saad Y, Schultz MH. Data communications in hypercubes. *Journal of Parallel and Distributed Computing* 1989; **6**: 115–135.
5. Saad Y, Schultz MH. Data communication in parallel architectures. *Research Report YALEU/DCS/RR-461*, Yale University, 1986.
6. Saad Y, Schultz MH. Topological properties of hypercubes. *Research Report YALEU/DCS/RR-289*, Yale University, 1985.
7. van de Geijn R. On global combine operations. *Journal of Parallel and Distributed Computing* 1994; **22**:324–328.
8. Ben-Miled Z, Fortes JAB, Eigenmann R, Taylor VE. On the implementation of broadcast, scatter and gather in a heterogeneous architecture. *HICSS'98*: *Proceedings of the Thirty-First Annual Hawaii International Conference on System Sciences*, vol. 3, Honolulu, HI, 1998; 216–225.
9. Goldman A, Trystram D, Peters J. Exchange of messages of different sizes. *Workshop on Parallel Algorithms for Irregularly Structured Problems*, Berkeley, CA, 1998; 194–205.
10. Gupta R, Balaji P, Panda DK, Nieplocha J. Efficient collective operations using remote memory operations on VIA-based clusters. *IPDPS '03*: *Proceedings of the 17th International Symposium on Parallel and Distributed Processing*, Washington, DC, U.S.A. IEEE Computer Society: Silverspring, MD, 2003; 46.2.
11. Huse LP. Collective communication on dedicated clusters of workstations. *Proceedings of the 6th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface*, London, U.K., 1999; 469–476.
12. Karonis NT, de Supinski BR, Foster I, Gropp W, Lusk E, Bresnahan J. Exploiting hierarchy in parallel computer networks to optimize collective operation performance. *IPDPS '00: Proceedings of the 14th International Symposium on Parallel and Distributed Processing*, Washington, DC, U.S.A. IEEE Computer Society: Silverspring, MD, 2000; 377–386.
13. Kielmann T, Hofman RFH, Bal HE, Plaat A, Bhoedjang RAF. MagPIe: MPI's collective communication operations for clustered wide area systems. *PPoPP '99*: *Proceedings of the SIGPLAN Symposium on Principles and Practice of Parallel Programming*. ACM: New York, May 1999; 131–140.
14. Tsai Y-J, McKinley PK. An extended dominating node approach to broadcast and global combine in multiport wormhole-routed mesh networks. *IEEE Transactions on Parallel and Distributed Systems* 1997; **8**(1):41–58.
15. Wang S-Y, Tseng Y-C, Ho C-W. Efficient single-node broadcast in wormhole-routed multicomputers: A network-partitioning approach. *SPDP '96: Proceedings of the 8th IEEE Symposium on Parallel and Distributed Processing*, Washington, DC, U.S.A. IEEE Computer Society: Silverspring, MD, 1996; 178.

DOI: 10.1002/cpe

16. Wu M-S, Kendall RA, Wright K, Zhang Z. Performance modeling and tuning strategies of mixed mode collective communications. *SC '05: Proceedings of the 2005 ACM/IEEE Conference on Supercomputing*, Washington, DC, U.S.A. IEEE Computer Society: Silverspring, MD, 2005; 45.
17. Seitz CL. The cosmic cube. *Communications of the ACM* 1985; **28**(1):22–33.
18. NCUBE Company. *NCUBE 6400 Processor Manual*.
19. Faraj A, Yuan X. Automatic generation and tuning of MPI collective communication routines. *ICS '05*: *Proceedings of the 19th Annual International Conference on Supercomputing*, New York, NY, U.S.A. ACM Press: New York, NY, 2005; 393–402.
20. Barnett M, Gupta S, Payne D, Shuler L, van de Geijn RA, Watts J. Interprocessor collective communication library (InterCom). *Proceedings of the Scalable High Performance Computing Conference 1994*, Knoxville, TN, 1994.
21. Barnett M, Littlefield R, Payne D, van de Geijn R. On the efficiency of global combine algorithms for 2-D meshes with wormhole routing. *Journal of Parallel and Distributed Computing* 1995; **24**:191–201.
22. Barnett M, Payne D, van de Geijn R, Watts J. Broadcasting on meshes with wormhole routing. *Journal of Parallel and Distributed Computing* 1996; **35**(2):111–122.
23. Chan EW, Heimlich MF, Purkayastha A, van de Geijn RA. On optimizing collective communication. *Proceedings of the 2004 IEEE International Conference on Cluster Computing*, San Diego, CA. IEEE: New York, 2004; 145–155.
24. Gropp W, Lusk E, Doss N, Skjellum A. A high-performance, portable implementation of the MPI message passing interface standard. *Parallel Computing* 1996; **22**(6):789–828.
25. Snir M, Otto S, Huss-Lederman S, Walker DW, Dongarra J. *MPI: The Complete Reference* (2nd edn), vol. 1, The MPI Core. The MIT Press: Cambridge, MA, 1998.
26. Thakur R, Rabenseifner R, Gropp W. Optimization of collective communication operations in MPICH. *International Journal of High-Performance Computing Applications* 2005; **1**(19):49–66.
27. Watts J, van de Geijn R. A pipelined broadcast for multidimensional meshes. *Parallel Processing Letters* 1995; **5**(2): 281–292.
28. Lillevik SL. The Touchstone 30 Gigaflop DELTA Prototype. *Proceedings of the Sixth Distributed Memory Computing Conference*. IEEE Computer Society Press: Silverspring, MD, 1991; 671–677.
29. Almasi G, Archer C, Castanos JG, Gunnels JA, Erway CC, Heidelberger P, Martorell X, Moreira JE, Pinnow K, Ratterman J, Steinmacher-Burow BD, Gropp W, Toonen B. Design and implementation of message-passing services for the Blue Gene/L, supercomputer. *IBM Journal of Research and Development* 2005; **49**(2/3):393–406.
30. Chan E, Gropp W, Thakur R, van de Geijn R. Collective communication on architectures that support simultaneous communication over multiple links. *Proceedings of the 2006 SIGPLAN Symposium on Principles and Practices of Parallel Programming*, New York, NY, U.S.A. ACM: New York, 29–31 March 2006; 2–11.
31. Payne D, Shuler L, van de Geijn R, Watts J. Streetguide to collective communication, unpublished manuscript.

DOI: 10.1002/cpe