on Multidimensional Meshes with Wormhole Routing

Jerrell Watts
Department of Computer Sciences
The University of Texas at Austin
Austin, Texas 78712-1188
jwatts@cs.utexas.edu

In Partial Fulfillment of the Requirements for the Computer Sciences Honors Program

May 6, 1994

Abstract

This work presents a methodology for constructing efficient collective communication libraries for mesh-based multicomputers with wormhole routing. Such libraries can be built from a small set of simple primitives. Specifically, two classes of routines are presented: short vector routines that minimize message startups and long vector routines that minimize data transmission. Routines from these two classes can then be hybridized, resulting in algorithms which are efficient over a wide range of machine performance and vector size parameters. Implementations of these basic routines are given, and the derivation of other routines from them is demonstrated. Furthermore, these methods are applicable to meshes in their full generality: There are no restrictions on the relative dimensions of the mesh or the number of nodes contained therein (e.g., they do not require power-of-two partition sizes). Also, the schemes presented here inherently support group commu-

hostile programming environment. Even those libraries currently offered by vendors are not yet standardized and often offer questionable performance [4]. Library interface standards have been proposed, but they do not specify the underlying methods used to achieve an efficient implementation [16]. To that end, this paper addresses the problem of actually building a high-performance communications library on mesh-based architectures with wormhole routing. Just as other standardized libraries such as the Basic Linear Algebra Subroutines (BLAS) must be based on architecture-dependent implementations for maximum performance, so must implementations of collective communications libraries take into account the vagaries of the target machines. Thus, it is the purpose of this thesis to provide the algorithmic underpinnings that will provide the same level of performance that a user would expect from hand-coded mathematical libraries.

This thesis targets eight collective communication operations which are found in many applications and which have been incorporated into library standards such as the Message Passing Interface (MPI) [16]. They are the *all-to-all scatter,* the *broadcast*, the *collect*, the *distributed reduction*, the *gather,* the *reduction-to-all,* the *reduction-to-one* and the *scatter.* These routines are defined as collective communications because all nodes either participate as contributors, receiv-

order pipelines. The same techniques used for one operation can be applied to routines with similar (possibly inverted) data flow patterns. In one such case, the same methods used for the broadcast are simply reversed to provide an efficient implementation of the reduction-to-one operation. Beyond this, at the level of the full routines themselves, there is also considerable re-use. For example, the long vector reduction-to-all operation is implemented using versions of the distributed reduction and collect operations. In this manner, the actual difficulty of library construction is greatly reduced by minimizing the amount of code that must be written and tested. These interdependencies have an additional impact: The benefits of a more efficient implementation of a particular operation extend beyond the operation itself to all of the other operations that are built on top of it.

Another primary distinction of this work is its intrinsic support for group-based communications. Specifically, the methods presented herein are obviously quite applicable to structured groups of nodes (e.g., groups that comprise rows, columns or blocks in the physical mesh). They also offer excellent support and performance for unstructured groups, with arbitrary physical layouts. In doing so, these techniques fully provide for the group-based collective communications strategy proposed by MPI.

unique label, $\mathbf{P}_0,\ldots,\mathbf{P}_{p\text{-}1}$.

2) The links between nodes are bidirectional. I.e., they can simultaneously transport a message in each direction. However, if more than one message traverses a link in either direction, they share the one-way bandwidth of that connection. Note, however, that the bandwidth of the network is usually greater than the bandwidth out of a node. As a result of this excess bandwidth, the actual performance penalty for contention may be less than expected. (See below.)

3) The interconnect uses wormhole routing. In such a network, communication is considered to be distance-independent.[1] Given this assumption, sending a message n data units long from one node to another under contention-free conditions requires time $\alpha + n\beta$, where α and β represent the message start-up time and per data item transmission time, respectively. Furthermore, wormhole routing implies that a message simultaneously occupies all of the channels on the path between the source and destination node, as determined by the routing method. In reference to

direction in each dimension. For example, on a two-dimensional mesh, if one were to restrict all communication to proceed “south” or “east,” all logical wrap-arounds from the last to first node within each column or row would go “north” or “west,” respectively, and would not conflict.

4) Messages are routed using dimensional-order or e-cube routing. In other words, a message is routed entirely through each dimension, starting with the lowest dimension and ending with the highest dimension. For example, on a three-dimensional mesh, a message would be routed through the *x*-dimension first, then the *y*-dimension, and finally the *z*-dimension.

5) The nodes are single-ported. Therefore, a node can simultaneously send a single message to and receive a single message from the same or different nodes with no time penalty. (If more than one message is being received at any given time, then Assumption 2 applies.)

6) The vendor-supplied communication library supports point-to-point message-pass-

$p_{i,j}$ is the j^{th} factor of m_i, the length of the i^{th} dimension.

3 Target Architectures

The algorithms and techniques presented in this paper are designed for machines with multidimensional mesh interconnects using wormhole routing. Commercial examples of such machines include the Intel Paragon and the Cray T3D, which utilize two-dimensional non-torus and three-dimensional torus meshes, respectively [7,12]. They are also appropriate for research machines such as the Intel Touchstone Delta (two-dimensional non-torus) and the J- and M-Machines (three-dimensional non-torus) [9,13,14,15,17]. Many of these algorithms and techniques also apply to hypercubes (which can be viewed as multidimensional meshes with two nodes in each dimension), although that architecture is not specifically addressed.

4 Target Collective Communications

The following collective communications routines are targeted by this paper::

Table 1: Summary of Target Collective Communications

Operation	Before	After

Reduction-to-all	$y^{(j)}$ at $\mathbf{P}_j$	$\oplus_{i=0}^{p-1} y^{(i)}$ at all $\mathbf{P}_j$, $\oplus$ given
Reduction-to-one[e]	$y^{(j)}$ at $\mathbf{P}_j$	$\oplus_{i=0}^{p-1} y^{(i)}$ at $\mathbf{P}_k$, k and $\oplus$ given
Scatter[f]	x at $\mathbf{P}_k$, k given	x_j at $\mathbf{P}_j$

a. Sometimes referred to as a *transpose* or *complete exchange*.
b. Sometimes distinguished as a *one-to-all broadcast*.
c. Sometimes referred to as an *all-to-all broadcast*.
d. Sometimes referred to as a *fan-in*.
e. Also (confusingly) referred to as a *fan-in*.
f. Sometimes referred to as a *fan-out*, *multicast* or *one-to-all personalized broadcast*.

5 Short Vector Primitives

When an operation is performed using short vectors, startup latencies are the primary time component. In these situations, the best approach is one that minimizes the number of message startups. On meshes, such algorithms are usually based on a minimum spanning tree or prime factors rings approach. In particular, the broadcast, scatter and gather operations are implemented using minimum spanning trees, and the collect, all-to-all scatter and reduction operations use prime factors rings as well as minimum spanning trees.

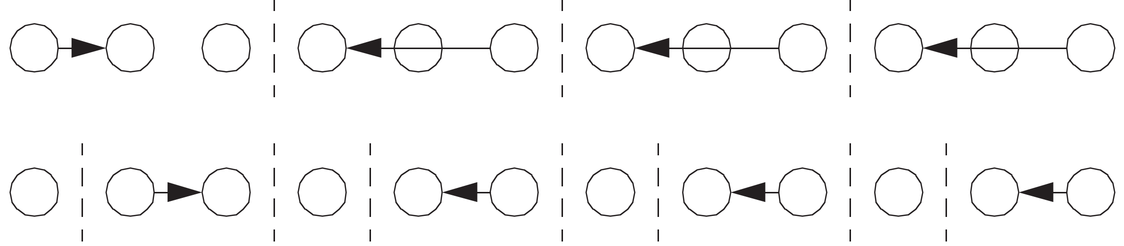
(a) (b)

Figure 1: An example of how the routing algorithm used in a mesh can create a contention. In both (a) the linear array and (b) the two-dimensional mesh, two messages are being sent/received by nodes in separate partitions. In the (a) this causes no problems, but in (b) the *x*-axis first routing strategy results in a conflict on one of the links.

approach requires that the mesh be of power-of-two size [3]. Second, it can generate tremendous message conflicts due to the fact that message paths that are disjoint on hypercubes may overlap on meshes [2]. As a first cut at a more general approach, one might try to partition the mesh in half at each stage, sending to or receiving from the first node in the other half. On a non-power-of-two size mesh, this approach can create network conflicts. (See Figure 1.) With a few modifications, one can avoid these problems.

The resulting recursive-splitting algorithm, which was first presented in [3], is as follows: Split the current partition into two approximately equal subpartitions. When the new subpartition is to the right (left) of the root node, its new root is the rightmost (leftmost) node in that subpartition. In the root node's subpartition, the algorithm repeats as specified above. If the new subpartition is to the right (left) of the root node, it also recursively divides its subpartition, but the new root is always the leftmost (rightmost) node in the new subpartition. (See Figure 2.)

MST based operations incur $\lceil \log_2(p) \rceil$ startups and are thus quite useful for distributing and


(a)

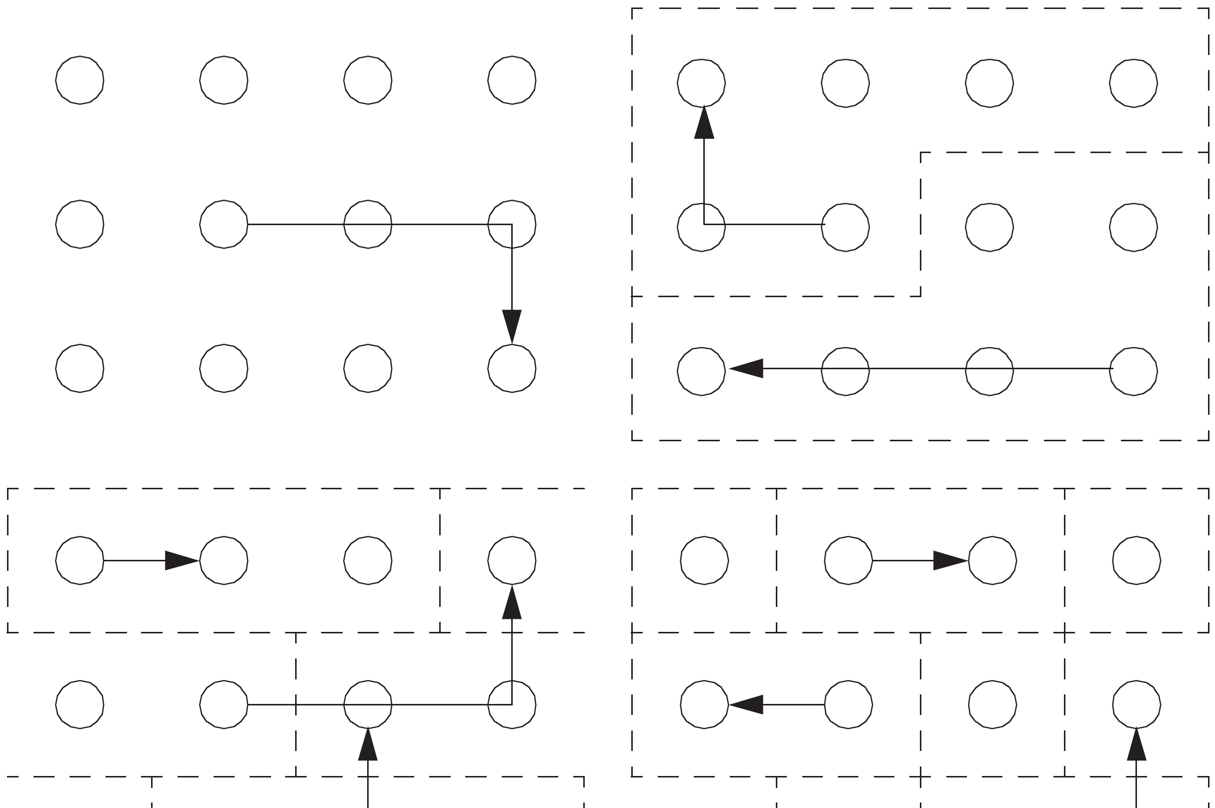


Figure 3: The PFR for a dimension containing six nodes.

5.2 Prime Factors Rings

Binary exchanges are a fundamental component of many hypercube algorithms, in which nodes trade data with their nearest neighbors in each dimension. Like the bit-toggling MST implementation, they are restricted to power-of-two size meshes. Prime factors rings (PFR) are a generalization of the binary exchange for arbitrary mesh sizes.

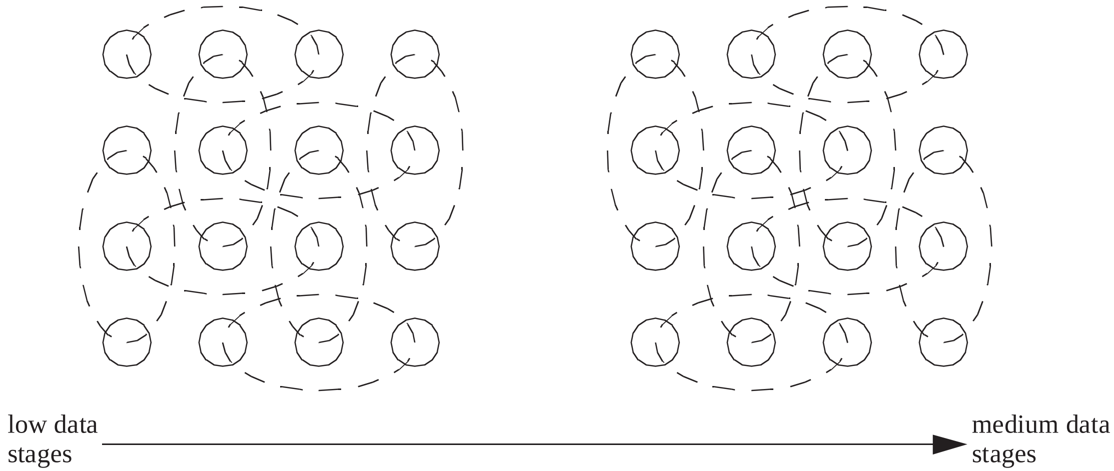
In using PFR, one must first calculate all of the prime factors (with repetitions) of the mesh size. For example, for $p = 60$, the prime factors would be 2, 2, 3 and 5. The nodes are then organized into rings, each of which contains a number of nodes equal to one of the prime factors. The nodes then circulate data within each ring of which they are a member. Notice that in the case of power-of-two size meshes, the prime factors are all two, and so the prime factors rings are simply binary exchanges between pairs of nodes. Therefore, this approach will require at least $\log_2(p)$ message operations. In the worst case, where m_i is a prime number for each dimension i, the PFR method will involve $m_0 + \mathrm{m}_1 + ... + m_{d\text{-}1} - d$ startups.

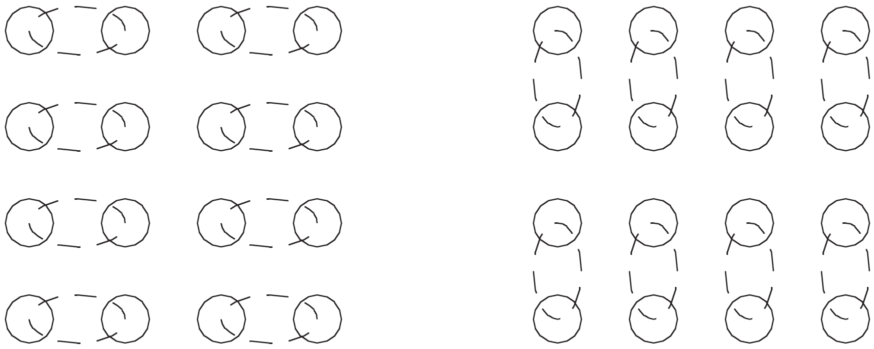
Just as with the MST technique, one must be careful in constructing PFR on a mesh. The best considerably by utilizing rings that lie in different dimensions in the mesh. By doing so, one effectively reduces contention by a factor of $1/d$ for large d-dimensional meshes. (See Figure 4.) In such cases, let $c_{i,j}$ be the number of active rings in the i^{th} dimension that are interleaved with the j^{th} ring of that dimension. Thus, there will be at most $c_{i,j}$ messages contending for a given link during a step of the PFR method. The alternation modification may complicate the algorithm somewhat since non-contiguous portions may be sent or received, requiring that the data be rearranged at the sending and receiving ends. One should also take care to structure the rings so that nodes are grouped most closely in the phases where the most data is transferred.

6 Long Vector Primitives

For long vectors, minimization of data transmission is essential. In these cases, extra message operations may be incurred if the volume of data can be significantly reduced. Primitives in this arena typically use a bucket approach, in which all of the nodes in single dimension form a ring and circulate pieces among themselves. Otherwise, they use a pipelined approach in which the data vectors are broken into pieces and their transmission times are overlapped by pipelining them throughout the mesh along edge-disjoint spanning trees. The former approach is applicable to the

(a)





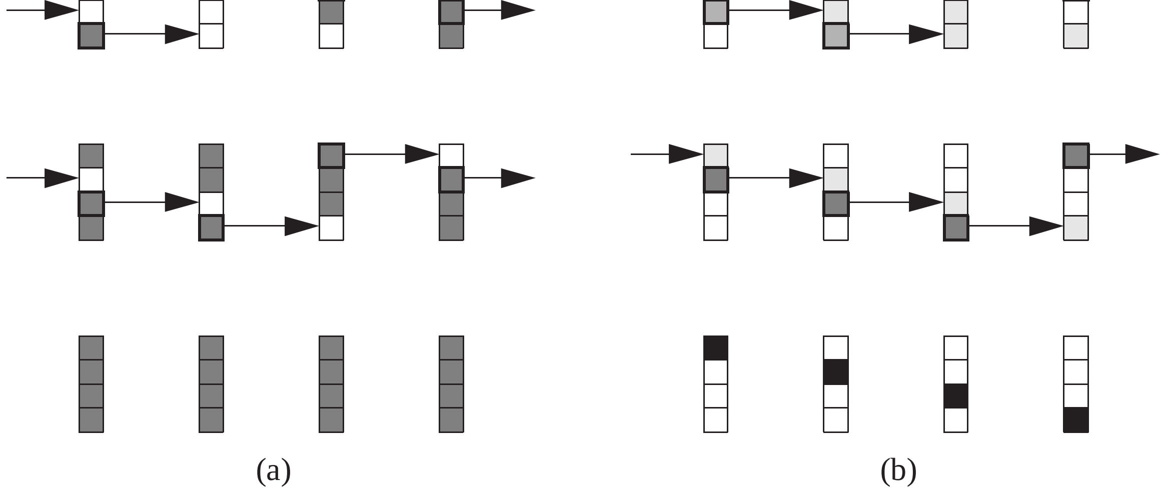


Figure 5: Dimensional rings. (a) shows the use of a dimensional ring to perform a collect. The darkened boxes indicate the portions that have been received thus far. In (b), the ring is used to perform a distributed reduction. In this case, the shade of a box represents a subvector's combination with other subvectors via the provided reduction operator.

6.1 Dimensional Rings

Since the PFR approach can generate considerable contention in a mesh where the dimensions' lengths are factorizable, it will perform very poorly for long vectors. In the dimensional rings (DR) scheme, the nodes are organized into rings comprising all of nodes in each dimension. For example, on a two-dimensional mesh, the nodes would be grouped into row rings and column rings. The nodes then circulate pieces within each ring. (See Figure 5.) While this approach has alternates between these trees, sending a packet along one then moving to the next one. The interactions of the packets as they move down the pipelines are such that a series of contention-free bidirectional exchanges occur in each dimension. There are severe problems, however, in generalizing the EDST approach to arbitrary meshes. This problem was addressed extensively in [24]. In short, the spanning trees of a mesh must be considerably deeper. While the trees of the EDST algorithm are only of $\log_2(p) + 1$ depth, edge-disjoint spanning trees for meshes governed by the assumptions of Section 2 are of depth $m_0 + m_1 + ... + m_{d-1} - d$. Furthermore, the nature of these spanning trees is such that they often produce traffic collisions as multiple messages arrive at particular nodes simultaneously. The resulting delays create “bubbles” in the message pipelines, and soon other nodes begin to receive multiple messages or none at all. Before long, the entire structure collapses, and the algorithm attains only a fraction of its expected performance.

The dimensional-order pipelined (DOPL) is quite simple: Instead of continuously alternating between dimensions, each part of the vector is completely pipelined along all the nodes in a dimension before proceeding to the next dimension. Doing so ensures that all of the nodes in a dimension are in the same phase. This, coupled with explicit synchronization, prevents a node of consecutive dimension positions that the “head” node of that row shares with the source node, starting with the (i mod d)th dimension. At the same time, the “tail” node in the row sends the (i - d)th piece directly to the “head” node in its row (unless the “head” node is the source node). If either $i + j - d$ or $i - d$ is less than zero, then the nodes involved do nothing during step i. If $i + j - d$ is greater than $k_{alt} - 1$, then the nodes involved re-transmit the ($k_{alt} - 1$)th piece. (See Figures 6a and 6b.) Note that this pipeline originates at the root node and extends to the leaves. To create a pipeline that originates at the leaves and culminates at the root node, simply vary i from $k_{alt} - d - 2$ down to 0 and reverse the direction of the message sends.

7 The Gather/Scatter Operations

The gather and scatter operations are fundamental routines that are used either alone or in the implementation of other, more complex routines [3,4]. In the gather operation, all of the nodes contribute variable-length portions which are gathered into a single vector on the specified node. In the scatter operation, the specified node divides a vector into variable-length portions and distributes them among the other nodes. Since these operations are inverses of one another, an effi-

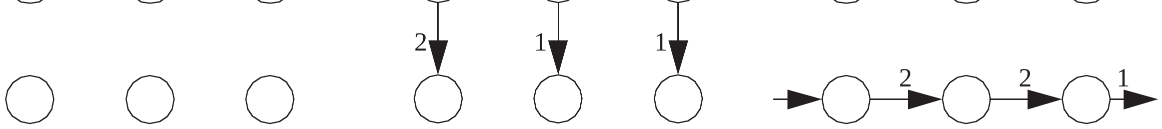
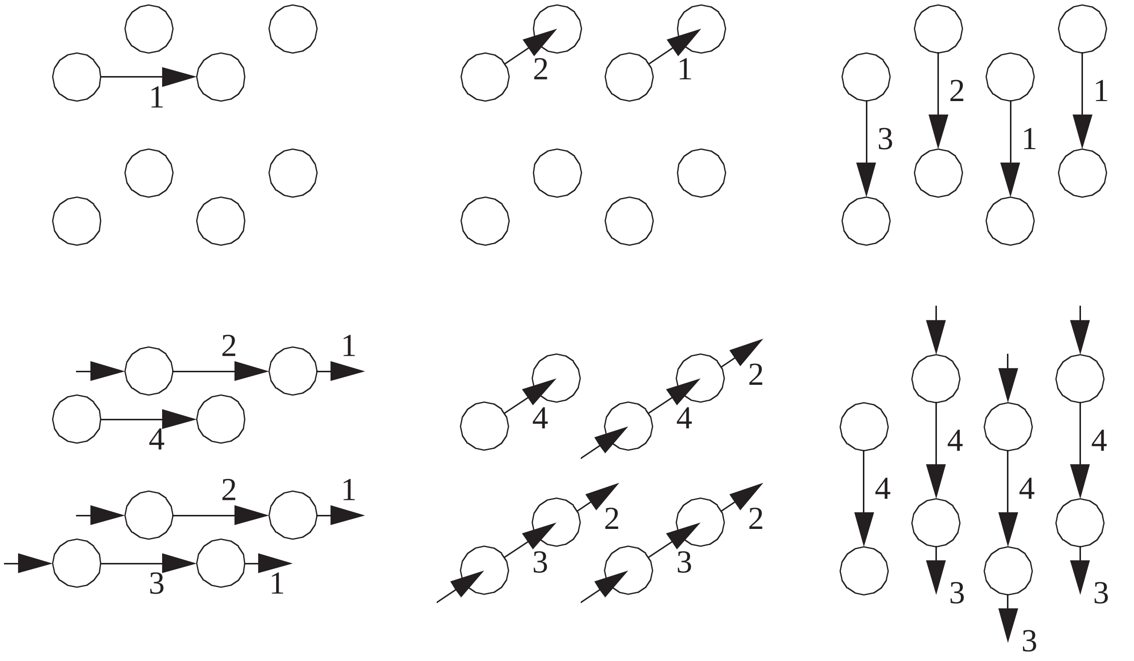


Figure 6a. The DOPL pipelines for a broadcast on a two-dimensional mesh with $k_{\text{alt}} = 2$ and the origin at the upper left. The arrows labeled 1 and 2 indicate, respectively, the channels over which the first and second half of the message are being pipelined. The only exceptions are the wraparound “1” sends on the far right; these are direct sends and are not pipelined.



to use a MST, with the parent node forwarding all of the data destined for the nodes in a subtree to the root of that subtree. The subtree's root would then proceed independently, sending to its subtrees and so forth.

In cases where the data is to be evenly distributed among the nodes, this technique performs very well, with a total time of:

$$T_{\text{MST-SCATTER}} = T_{\text{MST-GATHER}} = \lceil \log_2(p) \rceil \alpha + \frac{p-1}{p} n\beta$$

However, when the data distribution is uneven or skewed, this method can perform quite poorly. For example, consider the pathological case where one node is to receive the entire vector supplied by the root node. In this case, the running time would be:

$$\lceil \log_2(p) \rceil \alpha + n\beta \le T_{\text{MST-SCATTER}}, T_{\text{MST-GATHER}} \le \lceil \log_2(p) \rceil (\alpha + n\beta)$$

Thus, a better algorithm would partition the mesh by data distribution rather than partition it geometrically. In developing such an algorithm it is useful to state a necessary precondition for data flow optimality:

"In a data distribution-optimal scatter operation, the root node is the last to finish sending."

partition never exceeds the remaining transmission time for the root's partition, the root node will certainly finish last. By selecting the partitioning that *minimally* guarantees this, one also minimizes the number of message operations.

The resulting algorithm is exactly the same as the recursive-splitting algorithm presented in Section 5.1, except for the partitioning function. Instead of cutting the mesh in half, partition it so that the total data required by the nodes in the root's partition (not counting the root's own portion) is at least as much as the total data required by the nodes in the other partition (not counting the new root's portion).

It must be noted, however, that the above implementation is still not completely optimal. If the data distribution is skewed but the total data volume is small, the data distribution partitioning strategy may incur additional startups that are not offset by savings in data transmission time. Therefore, at each stage, if the imbalance in data distribution for the geometric partitioning drops below α/β bytes, it should be used instead of the data distribution partitioning, since any time savings in data transmission will be exceeded by the cost of an additional startup.

8 The Collect Operation

done with the last ring, they have the whole data contribution. Of course, there will be contention in all phases in which the nodes of a ring are not physically contiguous. Therefore, the early phases should use rings in which the nodes are most widely spaced, and the later stages rings where the nodes are closest. (See Figure 7.) The total running time is thus:

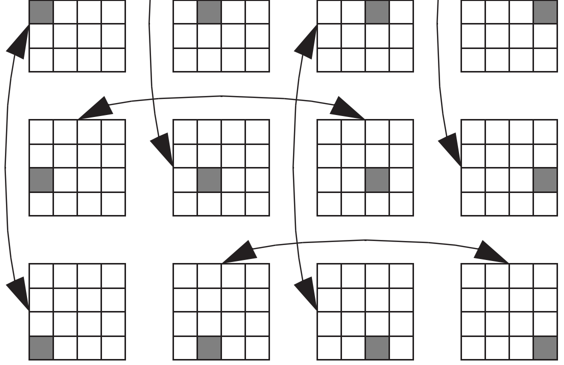
$$T_{\text{PFR-COLLECT}} = \sum_{i=0}^{d-1}\left(\sum_{j}(p_{i,j}-1)\left(\alpha + \frac{n}{\left(\prod_{i',j'} p_{i',j'}\right)}(\max(\beta_{\text{node}}, c_{i,j}\beta_{\text{net}}) + \delta)\right)\right)$$

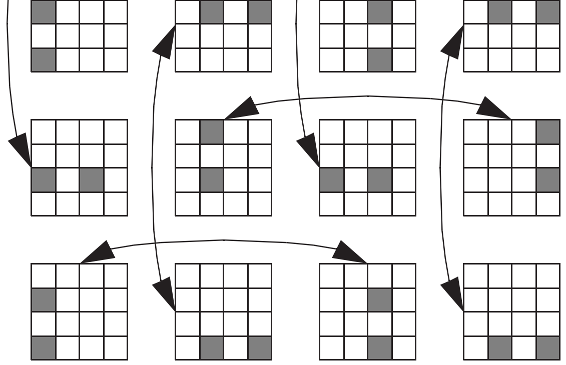
where j represents the rings in the i^{th} dimension and j' represents those rings that have not yet been collected over in the other d dimensions, i'. Note that all of the timing formulae in this section assume that the nodes' contributions are of uniform length.

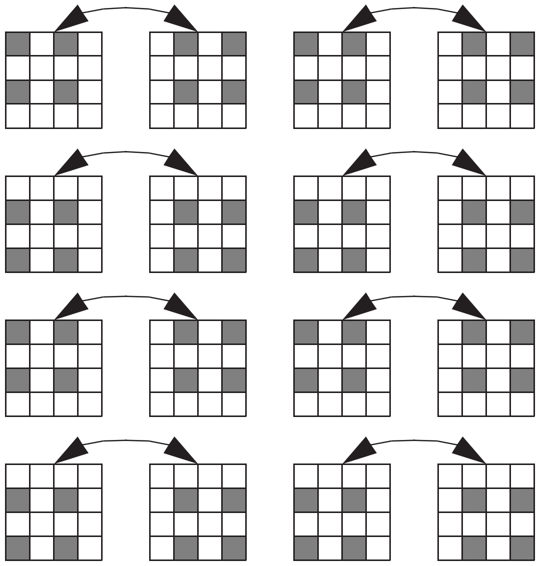
If the number of nodes does not factor well, a better approach would be to MST-gather the data to a single processor then MST-broadcast it to all of the nodes in the mesh. In this case, the time complexity is:

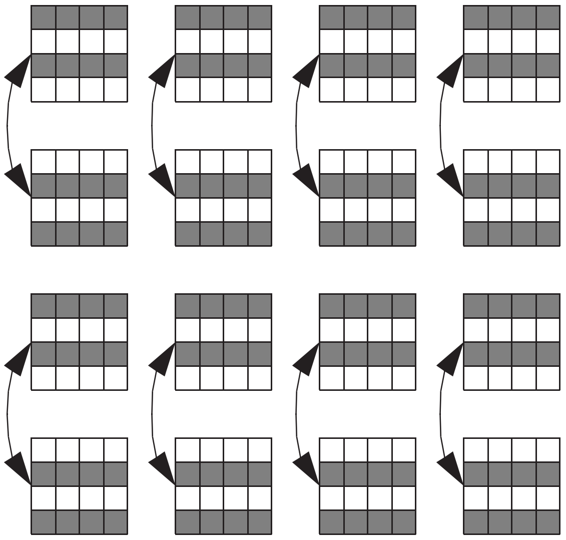
$$T_{\text{MST-COLLECT}} = T_{\text{MST-GATHER}} + T_{\text{MST-BROADCAST}}$$

For longer vectors, however, both of these approaches are clearly suboptimal since the total data transmission time is greater than the optimal $((p-1)/p)n\beta$. In such cases, the DR method is


(1)


(2)


(3)


(4)

enough to merit all of the startups required by the long vector method. A hybrid technique can be used to resolve this problem.

In the PFR-DR hybrid, one partially factorizes the dimensions. For example, on a three-dimensional mesh, one might choose to completely factor the first dimension, partially factor the second dimension and make a single ring of the third dimension. In this way, message startups are minimized in the initial stages of the collect when the vectors transmitted are small, and contention is minimized or eliminated in the later stages of the collect when the vectors are large. Note that the dimensional-alternation strategy for the PFR should only be used in the dimensions that are factorized; it should not be used in the dimensions containing pure DR. The time formula for this hybrid is:

$$T_{\text{PFR-DR-COLLECT}} = \sum_{i=d'}^{d-1} (m_i - 1)\left(\alpha + \frac{n}{\prod_{i'=i}^{d-1} m_{i'}}\beta\right) +$$

$$\sum_{i=0}^{d'-1}\left(\sum_{j}(p_{i,j} - 1)\left(\alpha + \frac{n}{\left(\prod_{i'=d'}^{d-1} m_{i'}\right)\left(\prod_{i'',j'} p_{i'',j'}\right)}(\max(\beta_{\text{node}}, c_{i,j}\beta_{\text{net}}) + \delta)\right)\right)$$

The MST implementation of a broadcast is quite simple; the entire vector is propagated along the MST given by the recursive-splitting algorithm. This approach yields optimal message startup times:

$$T_{\text{MST-BROADCAST}} = \lceil \log_2(p) \rceil (\alpha + n\beta)$$

While the MST-broadcast requires very few steps, it does involve a high data transmission cost since the entire vector in retransmitted $\lceil \log_2(p) \rceil$ times. The SC approach avoids this at the cost of additional message operations.

In the SC-broadcast, the vector is scattered though each dimension, starting with the highest dimension. It is then collected in each dimension, starting with the lowest dimension. All nodes then have the entire vector. For example, on a two-dimensional mesh, the vector is first MST-scattered equally among all the nodes in the source node's column. Each of these nodes then MST-scatters its portion to all of the nodes in its row. Then, the nodes then perform a DR-collect in rows and another in columns. The time for the entire operation is:

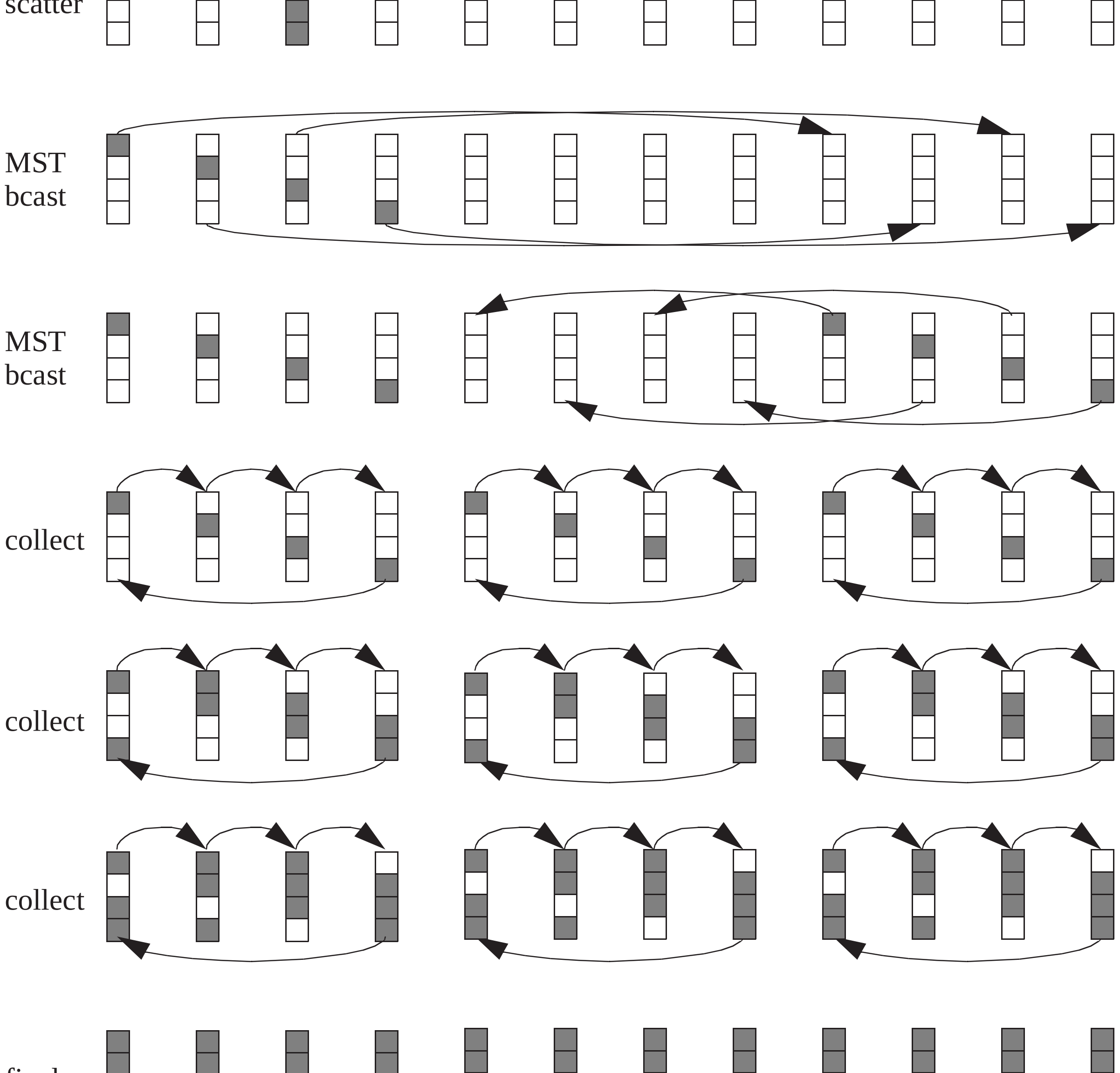
$$T_{\text{SC-BROADCAST}} = \left(\sum_{i=0}^{d-1} (\lceil \log_2(m_i) \rceil + m_i - 1) \right) \alpha + 2\frac{(p-1)}{p} n\beta$$

scattered through the real dimensions, MST-broadcasted within the virtual and remaining real dimensions, and then collected over the dimensions that it was initially scattered over. The nodes within the virtual dimension can either be physically contiguous or interleaved. If contiguous, the MST-broadcast phase is contention-free, but the collect phases create contention since groups of nodes are collecting across single links in the mesh. If interleaved, however, the collect phase, when the vectors are longer, is contention-free, and the short-vector MST-broadcast phase incurs the link contention. Thus, it is the later strategy that should be used in practice. (See Figure 7.) Using the interleaved MST-broadcast, the MST-SC hybrid has the following running time:

$$T_{\text{MST-SC-BROADCAST}} = \sum_{i=d'+1}^{d-1} \left(\lceil \log_2(m_i) \rceil + m_i - 1 \right) \left(\alpha + 2 \frac{n}{\prod_{i'=i}^{d-1} m_{i'}} \beta_{\text{node}} \right) +$$

$$\left(\lceil \log_2(p_{d',0}) \rceil + p_{d',0} - 1 \right) \left(\alpha + 2 \frac{n}{\left(\prod_{i'=d'+1}^{d-1} m_{i'} \right) p_{d',0}} \beta_{\text{node}} \right) +$$

$$\lceil \log_2(p_{d',1}) \rceil \left(\alpha + \frac{n}{\left(\prod_{i'=d'+1}^{d-1} m_{i'} \right) p_{d',0}} \max(\beta_{\text{node}}, p_{d',0} \beta_{\text{net}}) \right) +$$


scatter
MST bcast
MST bcast
collect
collect
collect

no contention in the MST phase.

For very long vectors, the pure SC broadcast is still less than optimal, so another approach must be used. The DOPL-broadcast, presented in [24], yields the best known time under the constraints specified in Section 2. The basic algorithm is to partition the vector into pieces, then pipeline it through each dimension, one dimension at a time. The total running time for the algorithm is:

$$T_{\text{DOPL-BROADCAST}} = (k_{\text{alt}} - 1)\left(\alpha + \frac{n}{k_{\text{alt}}}\beta\right)$$
$$+ \sum_{i=0}^{d-1}\left(\left\lceil \frac{k_{\text{alt}} + d - 1 - i}{d} \right\rceil (m_i - 1) + k_i - 1\right)\left(\alpha + \frac{n}{k_{\text{alt}} k_i}\beta\right)$$

where the first term represents the time required for the wraparound messages (if any), and the second term is the time required for the first *d* stages plus the pipeline fills throughout. k_i is the optimal partitioning in the ith dimension:

$$k_i = \begin{cases} 1 & \text{if } \sqrt{\dfrac{(m_i - 2)\, n\beta}{k_{\text{alt}}\alpha}} < 1 \\[2ex] n & \text{if } \sqrt{\dfrac{(m_i - 2)\, n\beta}{k_{\text{alt}}\alpha}} > n \end{cases}$$

tor sum of all the nodes' contributions. Note that this is logically the opposite of the collect, which *begins* with smaller contributions and *ends* with equal-length vectors on every node. Many of the same techniques used in Section 8 apply here as well.

On hypercubes, the standard procedure is to use recursive-halving, a variation of the binary exchange in which nodes trade and reduce half of their vector in the highest dimension, a quarter in the next lower dimension, and finally 1/*p* of their vector in the lowest dimension (assuming of course, that the distribution is even). Thus each node ends up with a complete reduction of 1/*p* of the contributed vectors. Uneven distributions or the use of the generalized PFR approach requires some additional complexity: In each ring, the nodes trade and reduce a portion equal in length to all the portions destined for the nodes in the remaining rings to which they belong. Finally, in the last ring, the nodes simply circulate and reduce their own portions. The execution time for the PFR method is:

$$T_{\text{PFR-DISTRIBUTED REDUCTION}} = \sum_{i=0}^{d-1}\left(\sum_{j}(p_{i,j}-1)\left(\alpha + \frac{n}{\left(\prod_{i',j'} p_{i',j'}\right)}(\max(\beta_{\text{node}}, c_{i,j}\beta_{\text{net}}) + \delta + \gamma)\right)\right)$$

where, as with the PFR-collect, j represents the rings in the i^{th} dimension and j' represents those

Neither of these methods work well for long vectors, however, since the first technique means considerable contention and the latter unnecessarily replicates data transmission and reduction operations. Once again, the solution is to use the DR technique. Just as in the PFR approach, the nodes start with the highest dimension and, at each stage, circulate and reduce all the portions of nodes in the lower dimensions. The time for this is:

$$T_{\text{DR-DISTRIBUTED REDUCTION}} = \left(\sum_{i=0}^{d-1} (m_i - 1) \right) \alpha + \frac{(p-1)}{p} n (\beta + \gamma)$$

Just as in the collect operation, the PFR and DR algorithms for the distributed reduction can be hybridized by partially factoring the dimensions to better balance the relative latency and transmission costs of the two methods. In this case, the operation's running time is:

$$T_{\text{PFR-DR-DISTRIBUTED REDUCTION}} = \sum_{i=d'}^{d-1} (m_i - 1) \left(\alpha + \frac{n}{\prod_{i'=i}^{d-1} m_{i'}} (\beta + \gamma) \right) +$$

$$\sum_{i=0}^{d'-1} \left(\sum_j (p_{i,j} - 1) \left(\alpha + \frac{n}{\left(\prod_{i'=d'}^{d-1} m_{i'} \right) \left(\prod_{i'',j'} p_{i'',j'} \right)} (\max(\beta_{\text{node}}, c_{i,j} \beta_{\text{net}}) + \delta + \gamma) \right) \right)$$

where d' is the lowest unfactored dimension, j represents the rings in the i^{th} dimension and j'

mind, the techniques used for the broadcast can be logically reversed and applied to the reduction-to-one.

The simplest way to perform a reduction-to-one is to use a MST. The operation proceeds much like a gather, with the nodes at the leaves sending their data vectors to the their parents, who combine the data with their own, using the user-supplied operation, and then pass the resulting vector to their own parents. This process continues until the root node has the reduction of all of the nodes' data. The time complexity is the same as that of a MST-broadcast, plus the time required to perform the user operation:

$$T_{\text{MST-REDUCTION-TO-ONE}} = \lceil \log_2(p) \rceil (\alpha + n(\beta + \gamma))$$

The reduction-to-one analogue to the SC technique for long-vector broadcasts is to perform a DR-distributed reduction followed by a MST-gather within each dimension until the entire result resides on the root node. (Note that this is the exact reverse of performing a scatter followed by a collect.) The time for this operation is a simple extension of the time for the SC-broadcast:

$$T_{\text{DG-REDUCTION-TO-ONE}} = \left(\sum_{i=0}^{d-1} (\lceil \log_2(m_i) \rceil + m_i - 1) \right) \alpha + \frac{(p-1)}{p} n (2\beta + \gamma)$$

$$T_{\text{MST-DG-REDUCTION-TO-ONE}} = \sum_{i = d' + 1}^{d-1} \left(\lceil \log_2 (m_i) \rceil + m_i - 1\right) \left(\alpha + \frac{n}{\prod_{i' = i}^{d-1} m_{i'}} (2\beta_{\text{node}} + \gamma) \right) +$$

$$\left(\lceil \log_2 (p_{d',0}) \rceil + p_{d',0} - 1\right) \left(\alpha + \frac{n}{\left(\prod_{i' = d'+1}^{d-1} m_{i'}\right) p_{d',0}} (2\beta_{\text{node}} + \gamma) \right) +$$

$$\lceil \log_2 (p_{d',1}) \rceil \left(\alpha + \frac{n}{\left(\prod_{i' = d'+1}^{d-1} m_{i'}\right) p_{d',0}} (\max(\beta_{\text{node}}, p_{d',0}\beta_{\text{net}}) + \gamma) \right) +$$

$$\left\lceil \log_2 \left(\prod_{i' = 0}^{d'-1} m_{i'} \right) \right\rceil \left(\alpha + \frac{n}{\left(\prod_{i' = d'+1}^{d-1} m_{i'}\right) p_{d',0}} (\beta_{\text{node}} + \gamma) \right)$$

where d' is the dimension in which the DG phase ends and the MST phase begins. As with the MST-SC hybrid, if this phase division occurs on a dimensional boundary, then d' is the lowest dimension in which the DG technique is used. Once again, $p_{d',0} = m_{d'}$ and $p_{d',1} = 1$. Similarly, there is no contention is this case.

Here, k_i is:

$$k_i = \begin{cases} 1 & \text{if } \sqrt{\dfrac{(m_i - 2)\, n\, (\beta + \gamma)}{k_{\text{alt}}\alpha}} < 1 \\ n & \text{if } \sqrt{\dfrac{(m_i - 2)\, n\, (\beta + \gamma)}{k_{\text{alt}}\alpha}} > n \\ \sqrt{\dfrac{(m_i - 2)\, n\, (\beta + \gamma)}{k_{\text{alt}}\alpha}} & \text{otherwise} \end{cases}$$

12 The Reduction-to-All Operation

The reduction-to-all operation applies the user-supplied operator to all the nodes' vectors, leaving the complete result on every node. A crucial routine in many parallel applications, efficient implementations of this algorithm have been studied in depth in [1,23].

For short vectors, there are two approaches that one could use. The first is a "fan-in/fan-out" method that in this thesis' terminology is a MST-reduction-to-one followed by a MST-broadcast. On machines such as the Intel Touchstone Delta, this technique has been shown to work well. However, the nodes on the Delta cannot simultaneously send and receive without penalty (a violation of Assumption 5 in Section 2). On more recent architectures such as the Intel Paragon or physically contiguous. As with other PFR approaches, this is minimized by using rings in alternating dimensions. In this case, however, there is no need to rearrange data since the entire vector is being used. The operation's time is thus:

$$T_{\text{PFR-REDUCTION-TO-ALL}} = \sum_{i=0}^{d-1} \left(\sum_{j} (p_{i,j} - 1) \ (\alpha + n\,(\max(\beta_{\text{node}}, c_{i,j}\beta_{\text{net}}) + \gamma)\,) \right)$$

where j represents i^{th} dimension's rings.

For longer vectors, a recursive-halving strategy proves the best. The standard PFR strategy involves considerable communication and computational redundancy since nodes transmit and operate on the same vectors. By cutting the vector in half at each stage, this redundancy is eliminated, since nodes transmit and operate on separate portions. Thus, the nodes essentially perform a distributed reduction followed by a collect. The time for this variation is:

$$T_{\text{PFR-DC-REDUCTION-TO-ALL}} = T_{\text{PFR-DISTRIBUTED REDUCTION}} + T_{\text{PFR-COLLECT}}$$

Once again, however, an important range of medium length vectors requires a hybrid approach for the best performance. The strategy which was given in [1] hybridizes the PFR-DC and PFR approaches. The distributed reduction is performed over a subset of the rings, leaving

$$T_{\text{PFR-PFR-DC-REDUCTION-TO-ALL}} =$$

$$\sum_{i=0}^{d-1}\left(\sum_{j}(p_{i,j}-1)\left(\alpha+\frac{n}{\left(\prod_{i',j''}p_{i',j''}\right)}(\max(\beta_{\text{node}}, c_{i,j}\beta_{\text{net}})+\delta+\gamma)\right)\right)+$$

$$\sum_{i=0}^{d-1}\left(\sum_{j'}(p_{i,j'}-1)\left(\alpha+\frac{n}{\left(\prod_{i',j}p_{i',j}\right)}(\max(\beta_{\text{node}}, c_{i,j'}\beta_{\text{net}})+\gamma)\right)\right)+$$

$$\sum_{i=0}^{d-1}\left(\sum_{j}(p_{i,j}-1)\left(\alpha+\frac{n}{\left(\prod_{i',j''}p_{i',j''}\right)}(\max(\beta_{\text{node}}, c_{i,j}\beta_{\text{net}})+\delta)\right)\right)$$

where j represents the rings over which the distributed reduction and collect phases are performed, j' represents the rings over which the PFR-reduction-to-all phase is performed and j'' represents the rings over which the distributed reduction (collect) has already been (has not yet been) performed in the d dimensions, i'.

For very long vectors, a DR-oriented approach yields optimal data flow and operation counts. Introduced as the "buckets" approach in [1], this approach utilizes a DR-distributed reduction followed by a DR-collect of the result. The running time is simply the sum of the times for these two operations:

Last but certainly not least is the all-to-all scatter. The all-to-all scatter is *the* data flow algorithm; every node sends an individual subvector to every other node in the mesh. On hypercubes, the all-to-all scatter can be performed quite easily, requiring only $\log_2(p)$ steps plus some additional data rearrangement time. Mesh-based implementations must contend with the much more restricted bisection bandwidth of the mesh architecture. As a result, a myriad of methods have been proposed, each of which have certain relative message latency, data transmission time and data rearrangement cost advantages [6,10,20]. Many of these techniques require power-of-two meshes or make poor use of excess network bandwidth (a fatal flaw as will be shown below). Thus, this thesis presents a generalization of the best of these techniques using the primitives previously outlined in Sections 5 and 6.

The interleaved binary exchange presented in [10] provides excellent performance for short- to medium-length vectors. Generalized to arbitrary mesh sizes by using the PFR primitive, the algorithm is as follows: At each stage, approximately $1/d$ of the nodes operate in rings that lie within each dimension. Inside these rings, the nodes circulate all the subvectors destined for nodes that are in the remaining rings of the other nodes of the current ring. In the final stage, the where *n* is the length of all of the vectors on all of the nodes (thus, 1/*p* of the vector resides on any given node) and *j* represents the rings in the *i*th dimension.

As an approach for long vectors, one might think to use the DR method. For example, on a two dimensional mesh, column rings would circulate subvectors until each node has all of its row's subvectors. The data on each node would then be rearranged so that each node's portion is contiguous. Finally, row rings would circulate subvectors until every node has all of its data. This method has a running time of:

$$T_{\text{DR-ALL-TO-ALL-SCATTER}} = \sum_{i=0}^{d-1} (m_i - 1)\left(\alpha + \frac{n}{2p}(\beta + \delta)\right)$$

Current trends in mesh architectures, expose weaknesses in the DR method, however. First, nearly every modern mesh architecture has considerable excess bandwidth, often a factor of two or more [4,14,18]. This allows one to get data to its destination much more directly at no penalty. Second, this technique underutilizes the network links. Although every node is sending and receiving, only the links within a single dimension are being used. Finally, message latencies are declining rapidly, and vendors are beginning to provide low-level "get" and "put" operations with incredibly low latencies. At the same time, bandwidth into the node is beginning to approach the requires a factor of $\sqrt{p}/4$ more on two-dimensional meshes) and instantly floods the network to capacity. On wormhole routed machines, this can be disastrous. When a packet is blocked in the network of such a machine, all of the channels it occupies are held locked, causing other packets to block, until the entire network becomes completely congested. Innovations in router design proposed in recent years could alleviate such a problem, however. In particular, the virtual flow control scheme introduced in [8] allows the channels occupied by a blocked packet to be used by other packets. This improvement decouples channel and flit buffer resources by providing multiple buffers for each channel. Virtual channel flow control is already used in architectures such as the Cray T3D as well as the J- and M-Machines [7,9,17]. When this modification becomes widely adopted, the direct-send method should perform quite well, with a running time of:

$$T_{\text{DIRECT-ALL-TO-ALL-SCATTER}} \approx (p-1)\,\alpha + \sum_{i=0}^{d-1} \frac{n}{2p}\max\left(\beta_{\text{node}}, \left\lceil \frac{m_i}{2} \right\rceil \beta_{\text{net}}\right)$$

For very long vectors, the direct send method is at least a factor of two better than the DR-all-to-all-scatter.

14 Group Communication

they can also be used to provide collective communications for user-defined groups by simply allowing the user to supply the input array of node ID's. Also, rather than writing separate routines for whole-mesh and group-based operations, one can instead write only the group-based routines. The whole-mesh operations simply call the group-based routines with a pre-defined group containing all of the nodes, a technique that has been adopted in MPI [16].

15 Implementation Details

When implementing the algorithms discussed in this thesis, there are a few considerations that should not be overlooked. In particular, the intricacies of hybrid selection, DOPL alternations, and group structure maintenance are worthy of further attention.

15.1 Hybrid Selection

Obviously, hybrids are useless unless there is an effective way to determine which strategy should be used for a particular mesh. One way to determine the optimal hybrid would be to find the minimum running time for all hybrids as well as the pure short and long vector approaches. While an exhaustive technique such as this will certainly produce the best strategy, the search may be it may prove useful to *cache* recent hybrid strategies used by a given operation. In many applications, the same operations are performed repeatedly on the equal size data structures. Under such circumstances, one could associate vector lengths with their strategies, re-using the strategies when the operation is performed again on a vector of the same (or nearly the same) length.

15.2 DOPL Alternations

On typical machines, k_{alt} will never exceed ten or fifteen, even for very long vectors. Given that the DOPL primitive should only be used for very long messages, the small search time for the optimal k_{alt} should be insignificant. The validity of this procedure can be ascertained from the derivative of timing function with respect to k_{alt}, which is monotonically increasing and equal to zero for some value of k_{alt}. While one cannot solve for this value of k_{alt}, one *can* tell that the time function is thus convex. Once the value of k_{alt} is calculated for a particular message length, it can then be retained for later re-use.

15.3 Group Structure Maintenance

When groups are created, it is essential to extract information about the physical layout of a group equal and that the product of those array lengths equals the number of nodes in the new group. If so, one can consider the group to be a $m'_0 \times m'_1 \times \ldots \times m'_{d-1}$ mesh, where m'_i is the length of the array for the i^{th} dimension. Often, if a group is particularly small or its layout cannot be ascertained, one can simply consider it to be a linear array.

16 Performance Data

Many of the techniques described in this thesis have already been put to use in the Interprocessor Collective Communications Library (InterCom) for the Intel Touchstone Delta and Paragon systems [4]. One notable exception is that InterCom does not include an all-to-all scatter operation (although one will be included in a future release). Also, it does not use the DOPL primitive for the broadcast and reduction-to-all operations. (In practice, the high message latencies found in the current operating systems of the target systems push the break-even point for the DOPL algorithm to such high vector lengths that its inclusion is not merited. However, low-latency architectures such as the Cray T3D as well as the J- and M-Machines could certainly benefit from such a technique.) Figures 8 through 12 show comparisons of the hybrid, long vector and short vector approaches for the broadcast, collect, distributed reduction, reduction to all and reduction to one

17 Conclusion

This paper proposes an effective methodology for building a high-performance collective communications library on a mesh-based parallel computer. The routines presented are built on a simple set of short and long vector primitives. Similarity between the routines is exploited by reapplying techniques used in other operations. Furthermore, many of the operations themselves are built using versions of other, simpler operations. In this manner code reuse is maximized and redundancy eliminated. Finally, the strategies used can be easily extended to support more advanced features of message-passing standards such as group operations.

Thus, this thesis does not merely present algorithms; it also proposes an overarching philosophy that defines the mechanics and interactions of those algorithms as they are combined in the implementation of an efficient collective communication library. Moreover, the applicability of this methodology extends beyond today's prevailing architectures; its inherently broad and adaptive nature make it appropriate for the mesh-based parallel machines of tomorrow as well as those of today.

Delta System operated by the California Institute of Technology on behalf of the Concurrent Supercomputing Consortium. Access to this facility was provided by the Intel Supercomputer Systems Division and the California Institute of Technology.