# *Broadcasting on Meshes with Worm-Hole Routing

Michael Barnett
Department of Computer Science
University of Idaho
Moscow, Idaho 83844-1010
mbarnett@cs.uidaho.edu

David G. Payne
Supercomputer Systems Division
Intel Corporation
15201 N.W. Greenbrier Pkwy
Beaverton, Oregon 97006
payne@ssd.intel.com

Robert A. van de Geijn
Department of Computer Sciences
The University of Texas at Austin
Austin, Texas 78712–1188
rvdg@cs.utexas.edu

Jerrell Watts
Scalable Concurrent Programming Laboratory
California Institute of Technology
Pasadena, California 91125
jwatts@scp.caltech.edu

Original Version: November 2, 1993
Revised Version: September 21, 1994

**Abstract**

We address the problem of broadcasting on mesh architectures with arbitrary (non-power-two) dimensions. It is assumed that such mesh architectures employ cut-through or worm-hole routing. The primary focus is on avoiding network conflicts in the various proposed algorithms. We give algorithms for performing a conflict-free minimum-spanning tree broadcast, a pipelined algorithm that is similar to Ho and Johnsson's EDST algorithm for hypercubes, and a novel *scatter-collect* approach that is a natural choice for communication libraries due to its simplicity. Results obtained on the Intel Paragon system are included.

## 1 Introduction

In this paper, we discuss the design of general purpose broadcast routines for mesh architectures like the Symult S2010, and the Intel Touchstone Delta and Paragon systems. These systems consist of a number of processing nodes connected by a communication network that employs worm-hole routing, thereby allowing a programming model that assumes all nodes are directly connected under contention-free conditions.

For most users of these machines, broadcasting a message from one node to all others is a matter of calling the appropriate library routine. On hypercubes, such a library routine often embeds a minimum spanning tree in the network, allowing the broadcast to complete execution in a time proportional to $\log_2(p)n$, where

*This research was sponsored in part by Intel SSD, the Intel Research Council, and the University of Texas Center for High Performance Computing.

$n$ is the vector length, and $p$ equals the number of nodes in the network. For hypercubes, better algorithms exist. In particular, if communication latency is ignored, asymptotically the cost of a broadcast can be reduced to be proportional to $n$, independent of $p$, by using Ho and Johnsson's EDST algorithm [5]. This algorithm is not widely used, probably because its complexity makes it less attractive for a library and difficult to modify for special cases.

The two-dimensional (2D) mesh architecture with wormhole routing is an attractive interconnection architecture for distributed-memory multicomputers. A mesh can be scaled to arbitrarily large configurations while retaining high link bandwidth. Moreover, the number of nodes in a mesh does not inherently need to be a power of two, in contrast with the hypercube. However, the advent of the mesh architecture has required the rethinking of some algorithms. Although the programming model for both hypercubes and meshes with worm-hole routing allow the user to assume total connectivity, many communication algorithms that do not incur network conflicts on hypercubes do incur such conflicts on meshes.

In this paper, we are concerned with the efficient implementation of broadcast algorithms on two dimensional meshes, concentrating on the long vector case, since this is when network conflict is most noticeable. With regard to this subject, the paper makes the following contributions:

- It introduces techniques for avoiding network conflicts in a minimum spanning tree based broadcast. While this broadcast is clearly inferior for long vector lengths on which we focus, the results are applicable to the short vector case as well as scatter and gather operations, for which minimum spanning tree algorithms are optimal.
- We give an introduction to pipelined broadcast algorithms for two dimensional meshes. We use a simple implementation to illustrate why pipelined broadcasts are not as appropriate for general purpose libraries as they are on hypercubes.
- We propose a new algorithm, the scatter/collect algorithm, which is a natural choice for libraries due to both its simplicity and performance.

We must emphasize that we purposely focus on the *techniques* more than pushing the technology to the limit. It is our opinion that one reason the minimum spanning tree algorithm is still popular is largely due to a failure to make papers on communication algorithms more accessible to computational scientists who use high performance parallel computers.

## 2 Assumptions

The target architectures for our algorithms are distributed-memory mesh multicomputers, including Multiple Instruction Multiple Data (MIMD) machines such as the Symult S2000, and the Intel Touchstone Delta and Paragon systems.

For our theoretical analysis, we use the following model:

1. The multicomputer consists of $p$ nodes, labeled $\mathbf{P}_0, \ldots, \mathbf{P}_{p-1}$.

2. The nodes physically form an $r \times c$ two-dimensional grid.

3. Communication is with only one node at a time; i.e., *multi-cast* communication is not implemented in hardware.

4. A node can send and receive simultaneously to and from the same or different nodes with no penalty.

5. If no network conflicts occur, exchanging messages of length $n$ between two nodes requires time $\alpha + n\beta$, where $\alpha$ and $\beta$ represents the communication startup time and per item (byte) transfer time.

6. A message between two nodes occupies the entire path between the two nodes. The logical path from node $\mathbf{P}_i$ to $\mathbf{P}_j$ is denoted by $\langle i, j \rangle$. The physical path is denoted as $[i, ..., j]$, listing the physical nodes the path is routed on. On a linear array or sub-mesh, the logical path and the physical path coincide: $\langle i, j \rangle$ refers to both of them.

7. Links between nodes can carry one message at a time in each direction. If more than one message traverses a link in the same direction, they equally share the bandwidth of that connection.

8. Splitting and concatenation of vectors does not consume any time.

Under this model, the following observation is immediate:

**Theorem 1** *Provided $p > 1$, the lower bound on latency for broadcasting is $\log(p)\alpha$, and the lower bound on transmission time is $n\beta$.*

**Proof:**

1. By Assumption 3, at each step, the number of nodes to which a data item has been sent can at most double. Hence the minimum number of steps is bounded below by $\log(p)$.

2. By Assumption 4 and 7 and the definition of broadcast, if $p > 1$, all data must leave the root node at least once.

$\square$

## 3 Minimum Spanning Tree Broadcasting

The most popular broadcast algorithm is based on embedding a minimum spanning tree from the node that originates the broadcast (the root) to all other nodes.

## 3.1 Naive hypercube-like mesh algorithm

Assuming both $r$ and $c$ (and hence $p$) are powers of two, and that $\mathbf{P}_i$ is the root, this broadcast *on hypercubes* forwards messages using the following algorithm: View the index of a node as a binary number. Initially, only the root owns the message. For $i = 1, \ldots \log(p)$, at step $i$ of the algorithm, all nodes $\mathbf{P}_j$ that already own the message send the message to node $\mathbf{P}_k$ where $k$ differs from $j$ only in the $i$th binary digit. The algorithm requires time $\log(p)(\alpha + n\beta)$ to complete on hypercubes, since no network conflicts occur. While this algorithm still works on power of two meshes, Fig. 1 shows how it does incur network conflicts. Indeed, the total time is now given by

$$T_{\text{naive}} = \sum_{i=1}^{\log(r)} \left[\alpha + \mathbf{2^{(i-1)}} n\beta\right] + \sum_{i=1}^{\log(c)} \left[\alpha + \mathbf{2^{(i-1)}} n\beta\right] = \log(p)\alpha + (r + c - 2)n\beta \qquad (1)$$

## 3.2 Conflict-free hypercube-like mesh algorithm

A simple reordering of the steps allows all network conflicts to be avoided on power of two meshes: For $i = 1, \ldots, \log(p)$, at step $i$ of the algorithm, all nodes $\mathbf{P}_j$ that already own the message send the message to node $\mathbf{P}_k$ where $k$ differs from $j$ only in the $(\log(p) - i + 1)$st binary digit. The algorithm requires time $\log(p)(\alpha + n\beta)$ to complete on meshes (and hypercubes), since no network conflicts occur, as is illustrated in Fig. 2. Note that in the special case of a power of two mesh, the broadcast first operates as a broadcast within the root's column, followed by independent broadcasts within rows. Each of these separate broadcasts is within a linear array, for which it is easy to see that our method avoids network conflicts. A more formal proof of this observation can be found in [1].

## 3.3 MST Broadcast on non-power-two meshes

But meshes, unlike hypercubes, are not restricted to contain a number of nodes that is a power of two. An examination of the better algorithm above, and Fig. 2, shows that this broadcast can be reformulated in the following way: View the nodes as a linear array. At each step, the linear array is divided in half, and the root sends a message to the corresponding node in the half of the linear array that doesn't contain the root. Next, both these nodes become roots for recursive broadcasts within the two separate halves of the network. The obvious extension to non-power of two meshes is to modify this alternative formulation by dividing the nodes in half as closely as possible and then proceeding as before. However, it is no longer the case that we can guarantee that this broadcast does not incur network conflicts on meshes. In particular, it is no longer the case that by viewing the mesh as a linear array the broadcast automatically occurs within groups of nodes that physically form a linear array.

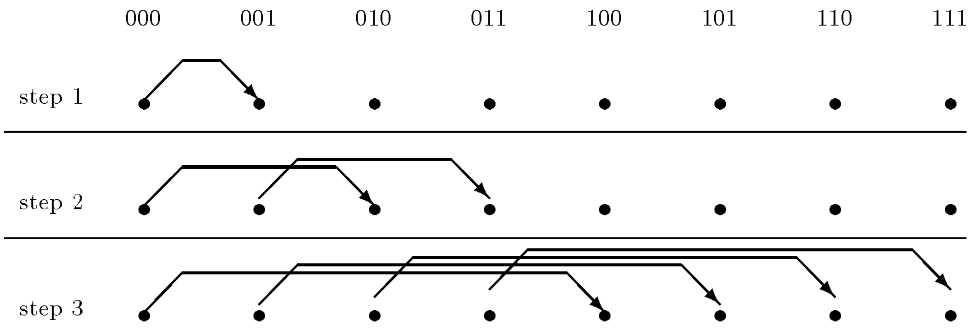



Figure 1: Hypercube MST broadcast viewing nodes as a linear array.

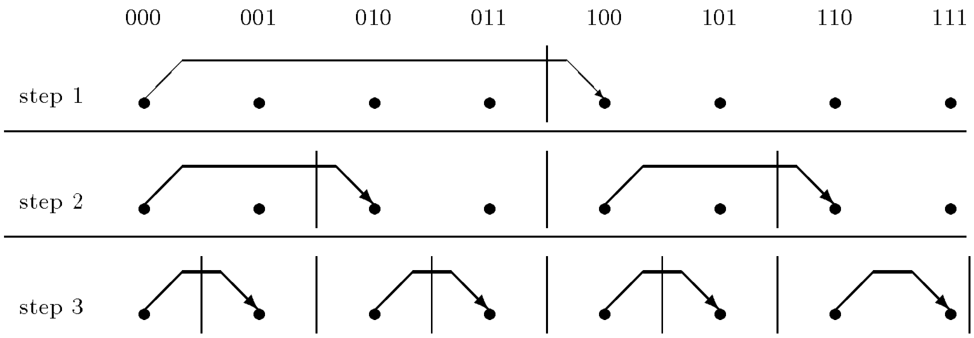



Figure 2: MST broadcast for nodes viewed as a linear array, using alternative order of steps.

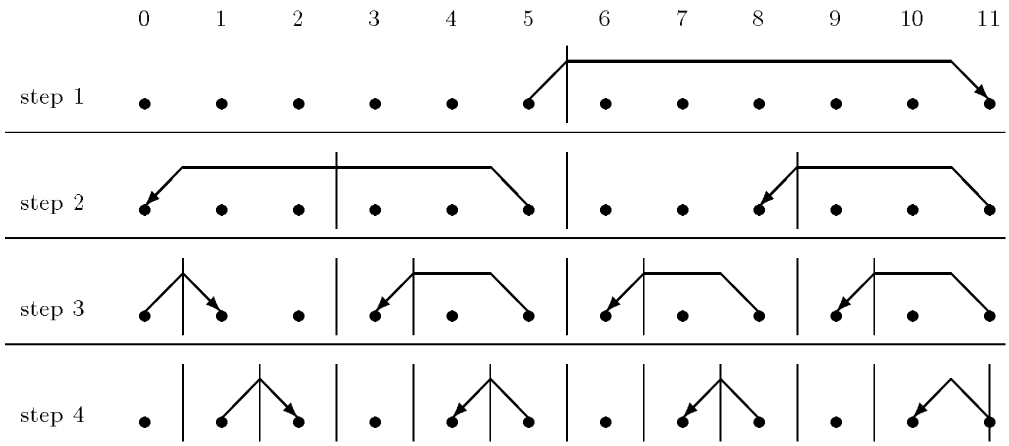



Figure 3: Recusive splitting broadcast for nodes viewed as a linear array. At each step, the node to which the message is sent is chosen so that subsequent communication is always in the direction of the original root.

### 3.3.1 Separating dimensions

To obtain an algorithm for meshes that avoids all network conflicts, we could formulate the broadcast as a broadcast within rows, followed by independent broadcasts within columns. However, this can require one additional step, since the time then becomes

$$\lceil \log(r) \rceil (\alpha + n\beta) + \lceil \log(c) \rceil (\alpha + n\beta)$$

Instead, we will carefully examine the kinds of communications that can generate network conflicts and use this knowledge to design a better minimum spanning tree broadcast for general meshes.

### 3.3.2 Avoiding network conflicts on general meshes

Consider the possible communication patterns generated by our minimum spanning tree broadcast that avoids network conflicts on linear arrays. All messages are in disjoint partitions (of the logical linear array) so no two messages overlap. Within the same step, any pair of messages is between nodes $i$, $j$, $k$, and $l$ such that $i < j < k < l$. There are four possibilities:

$$\langle i, j \rangle \quad \text{and} \quad \langle k, l \rangle \qquad \text{(Both to the right)} \tag{2}$$

$$\langle i, j \rangle \quad \text{and} \quad \langle l, k \rangle \qquad \text{(Both to the left)} \tag{3}$$

$$\langle j, i \rangle \quad \text{and} \quad \langle l, k \rangle \qquad \text{(Towards each other)} \tag{4}$$

$$\langle j, i \rangle \quad \text{and} \quad \langle k, l \rangle \qquad \text{(Away from each other)} \tag{5}$$

While it may seem counter-intuitive, given the simple routing schemes discussed in Lemma 2 below, it is the pair of messages $\langle j, i \rangle$ and $\langle k, l \rangle$, messages that are moving *away* from each other in the logical array, that can conflict on a network that is physically a mesh. This is demonstrated in Figure 4 for an *x-direction first* routing scheme. We show that the other three patterns (2)–(4) do not create any conflict under reasonable assumptions about the routing algorithm. This is summarized by the following lemma, which uses the same variables as in (2)–(4).

**Lemma 2** *Assume the routing algorithm for the network is such that $\langle i, j \rangle$ takes the shortest path, changing direction at most once,* x*-direction first. Then the physical paths used for the logical paths (2)–(4) do not conflict.*

**Proof:** See Appendix A. □

Thus, an algorithm that creates only the patterns described by (2)–(4) can be used to extend the conflict free hypercube-like algorithm to arbitrary 2D-meshes without incurring network contention.

### 3.3.3 Recursive Splitting Broadcast

We use these observations to generate the desired MST broadcast for non-power-two meshes by restating the algorithm described above: Again, view the nodes as a linear array. At each step, the linear array

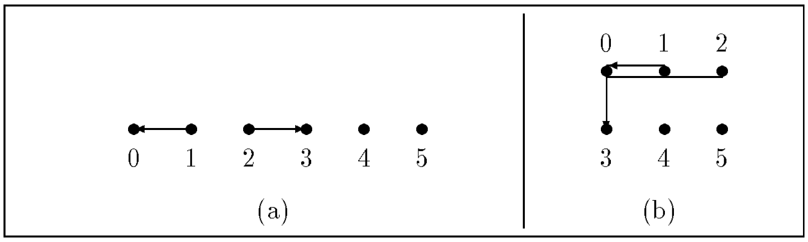



Figure 4: Example of the creation of conflict dependent on the routing algorithm and the mapping of the linear array to the mesh. (a) The paths $\langle 1, 0\rangle$ and $\langle 2, 3\rangle$ do not conflict. (b) The logical path $\langle 1, 0\rangle$ is routed on the physical path $[1, 0]$, while the logical path $\langle 2, 3\rangle$ is routed on the physical path $[2, 1, 0, 3]$ inducing conflict on the link between 0 and 1.

is divided approximately in half, and the root sends a message to a node in the half of the linear array that doesn't contain the root. Next, both these nodes become roots for recursive broadcasts within the two separate halves. We emphasize that for the correctness of the algorithm, the choice of which node to send to is arbitrary. By sending it to the node that is as far away (w.r.t. the logical linear array) from the root as possible, and making the restriction that in the half that does not contain the original root the broadcast must be staged to always generate messages that flow towards the original root (see Fig. 3), we can guarantee that all pairs of messages satisfy (2)-(4). We call this new algorithm the *recursive splitting* broadcast (RSbcast).

We formally state the properties of the recursive splitting broadcast in the following lemmas:

**Lemma 3** *All messages generated by RSbcast are sent within disjoint partitions of the logical array or are disjoint in time.*

**Proof:** This is by construction of the algorithm. Once a node sends a message, two partitions are created, and the broadcast is continued in each partition independently on a disjoint set of nodes. So a message is sent by a node within the same partition only after it has sent a message out of the partition in a previous step. □

**Lemma 4** *Any pair of messages generated by RSbcast that are not disjoint in time satisfy the conditions given by Equations (2)–(4).*

**Proof:** By Lemma 3, any pair of messages generated by RSbcast that are not disjoint in time are in disjoint partitions of the logical array. Let $\mathbf{P}_r$ be the original root of the broadcast. Define a node $\mathbf{P}_i$ to be a L-node or R-node if $i < r$ or $i > r$, respectively. We note that by construction the RSbcast has the property that any message originating from a L-node will move toward the right, while a message originating from a R-node will move toward the left. Moreover, a message that originates at a L-node or R-node is sent to a L-node or R-node, respectively. As a result, any pair of messages that don't originate at the same node has the property that:

| Case | Satisfies Equation |
|---|---|
| both originate at L-nodes | (2) |
| both originate at R-nodes | (3) |
| one originates at L-node, one at R-node | (4) |
| one originates at L-node, one at root | (2) or (4) |
| one originates at R-node, one at root | (3) or (4) |

We conclude that only the possibilities given by Equations (2)-(4) are encountered. □

An implication of the above lemmas is that if the message routing algorithm is chosen appropriately, no network conflicts will occur during the execution of the RSbcast algorithm, *even if the steps of the method are not perfectly synchronized.* We summarize our results in the following theorem.

**Theorem 5** *If the message routing algorithm is as given in Lemma 2, then the RSbcast algorithm proceeds without network conflicts, and the broadcast completes in $\lceil \log_2(p) \rceil$ steps, requiring time*

$$T_{\mathrm{rs}} = \lceil \log(p) \rceil (\alpha + n\beta). \tag{6}$$

An interesting observation is the fact that for both the MST broadcast on power of two meshes and the recursive splitting broadcast, it is not required to know the row and column dimensions of the physical mesh.

# 4 Pipelined Broadcast

For short messages, latency is often a dominating factor, which means an algorithm with the fewest steps (and thus the fewest messages) is appropriate. For long messages, however, the RSbcast performs poorly because the entire message is retransmitted $\lceil \log_2(p) \rceil$ times. Other techniques allow one to reduce the transmission time component, as we shall see next.

## 4.1 Pipelining on a Linear Array

When the $p$ nodes are arranged as a linear array, a broadcast from $\mathbf{P}_0$ can be accomplished by partitioning the message into $k$ equal-sized packets and pipelining them along the array [10]. Specifically, the root begins by sending the first packet to the second node in the array. At the same time the root node sends the next packet, the second node forwards the first packet to the next node, overlapping the transmission time of the two packets. The broadcast continues in this fashion, with each node receiving the next packet and forwarding the previous packet, until all of the packets have filtered through the array. The time for completion becomes:

$$T_{\text{pipe}} \quad = \quad (p-1)(\alpha + \frac{n}{k}\beta) + (k-1)(\alpha + \frac{n}{k}\beta) \tag{7}$$

The first term reflects the time for the first packet to reach the end of the array; the second term is the time for receiving the remainder of the packets. The optimal $k$, which is determined by taking the derivative of Equation 7 and solving for zero, is equal to:

$$k_{\text{opt}} = \begin{cases} 1 & \text{if } \sqrt{(p-2)n\beta/\alpha} < 1 \\ n & \text{if } \sqrt{(p-2)n\beta/\alpha} > n \\ \sqrt{(p-2)n\beta/\alpha} & \text{otherwise} \end{cases}$$

In practice, we round $k_{\text{opt}}$ to the nearest integer.

As discussed in Section 3, the linear array can be embedded in a mesh. The row major ordering of the nodes combined with the conditions of Lemma 2 guarantees that there will not be network conflicts in the pipe.

### 4.2 Pipelining on Hypercubes

For large meshes and hypercubes, the pipeline fill time is prohibitive. On machines with reasonable $\alpha$ and $\beta$, the array pipeline technique outperforms simpler strategies only for unreasonably long messages. On hypercubes, the pipeline depth is reduced considerably by embedding $\log_2(p)$ edge-disjoint minimum spanning trees rooted at the nearest neighbors of the source node. The source node alternates between the trees, sending a packet along each in round-robin fashion. This is the EDST broadcast first proposed by Ho and Johnsson [5]. The resulting pipe depth becomes $\log_2(p) + 1$.

The EDST algorithm is unsuitable for mesh architectures, however. In particular, the hypercube trees are no longer edge-disjoint when embedded in a mesh. The overlapping pipelines create considerable contention, destroying the performance of the algorithm. Moreover, the EDST algorithm inherently requires a power of two size mesh.

### 4.3 Pipelining on Meshes

When designing pipelined algorithms for mesh architectures, it is important to restrict communication to nearest neighbors in the physical mesh, in order to avoid undue network conflicts. The EDST strategy can be generalized to meshes by using two edge-disjoint "fences[1]" and alternating between them in the same manner as above. This technique was independently discovered by Bermond, Michallon and Trystram and is more thoroughly described in [4]. In that paper, the assumptions are somewhat different. Specifically,

[1]We use the term *fence* since we discovered the algorithm by first considering a method that creates paths that look like a fence, similar to techniques discussed in [8]

the authors assumed the mesh was a multi-ported, store-and-forward torus, rather than a single-ported, worm-hole routed mesh.

For simplicity, assume the root of the broadcast to be node $\mathbf{P}_0$. The nodes are coded in checkerboard fashion; the protocol is that black nodes send east and south during even and odd steps, respectively, while white nodes alternate in the opposite order. As a result, packets are forwarded along two edge-disjoint fences as illustrated in Fig. 4.3.

The root alternates between sending packages east and south, filling two pipelines of length $r + c$. Total execution time becomes:

$$T_{\text{edf}} = (k + r + c)(\alpha + \frac{n}{k}\beta) \tag{8}$$

with:

$$k_{\text{opt}} = \begin{cases} 1 & \text{if } \sqrt{(r+c)n\beta/\alpha} < 1 \\ n & \text{if } \sqrt{(r+c)n\beta/\alpha} > n \\ \sqrt{(r+c)n\beta/\alpha} & \text{otherwise} \end{cases}$$

(Even though the pipelines are of length $r + c$, synchronization needed during the algorithm forces two "empty" steps, resulting in $r + c + k$ steps instead of $r + c + k - 2$ steps.)

For the general root, the worm-hole routing property can be used to create a logical unidirectional torus, in which case the algorithm proceeds with the origin of the torus shifted to the actual root. That is, since all sends are either east or south in a single dimension, the wrap-arounds move west or north, respectively, and do not conflict with other messages.

The effective length of the pipe is within a constant of optimal for a uni-directional torus since a message must traverse a minimum of $r + c - 1$ links to get from the root to the most distant node.

# 5 Alternative Algorithm: Scatter-Collect

Ideas from our previous work on performing the global combine can be used to obtain an alternative tradeoff between the startup cost and the transfer cost [3]. We first present a simple algorithm for one-dimensional meshes, and then extend it for the two-dimensional case.

## 5.1 1-D scatter-collect

The RSbcast algorithm can be modified by splitting the vector in half at each step of the algorithm (the "scatter"). This leaves the vector distributed across the nodes, with each node possessing a piece of the original vector. A ring is then logically embedded in the nodes, and the pieces are circulated until all of the nodes possess all of the original vector (the "collect"). The algorithm is depicted in Figure 6. It shows the initial state with node 0 as the source for the broadcast, followed by $\lceil \log_2(p) \rceil$ steps for the scatter phase, and $p-1$ steps for the collect phase. Some of communications during the collect phase are redundant in that

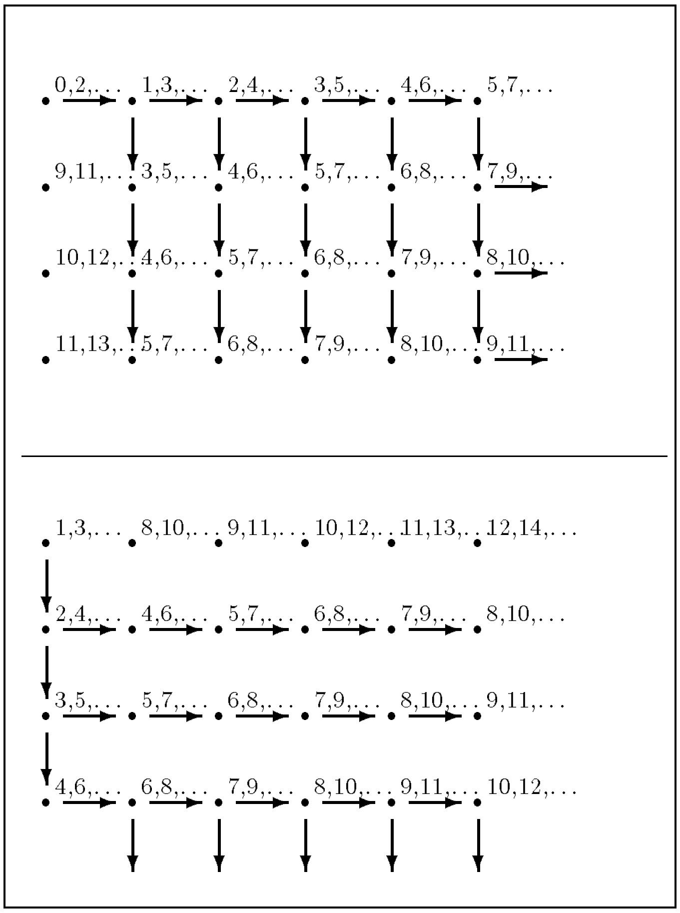



Figure 5: Pipelined broadcast on Edge Disjoint Fences. Two fences are embedded as given above. The root alternates sending packets using the top and bottom fences. The notation $i, i+2, \ldots$ is used to indicate that the first packet sent through the given fence arrives at the node at time $i$, followed by another packet every two time steps.

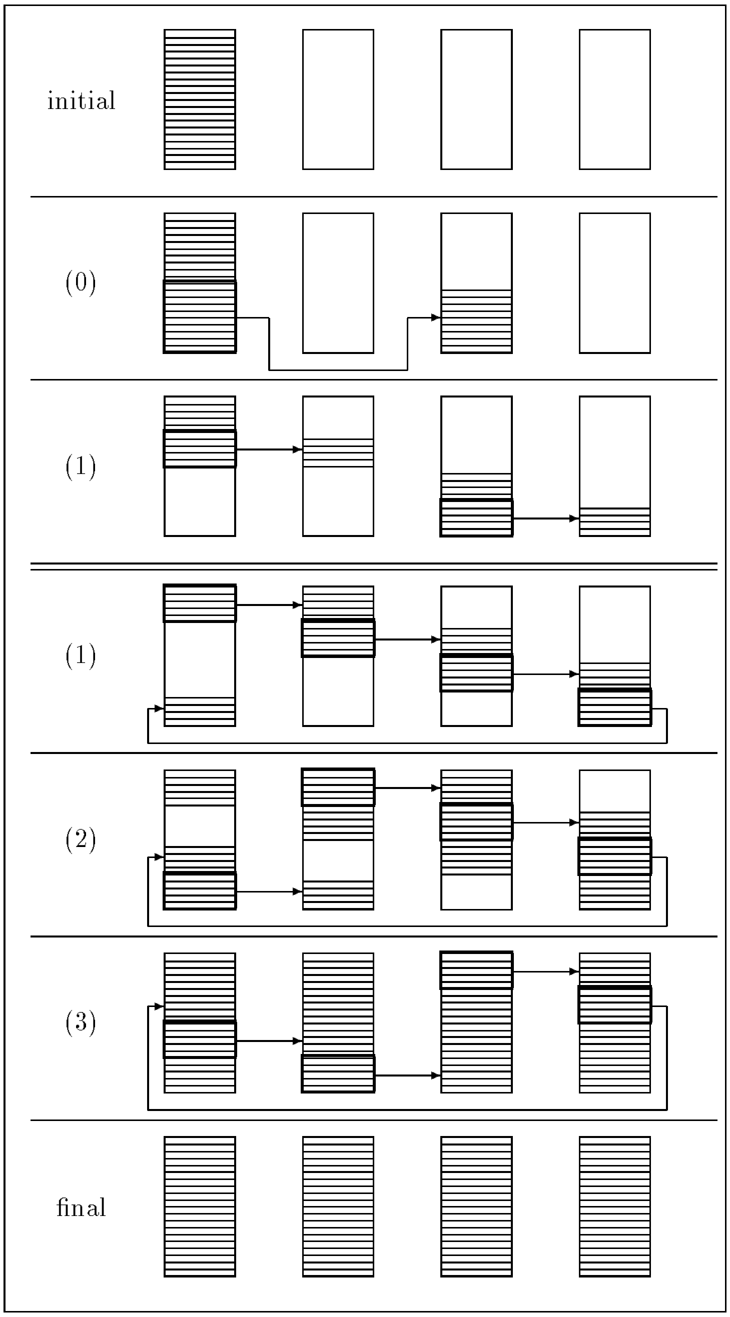



Figure 6: Scatter-collect for $p = 4$ with source 0.

nodes receive pieces of the vector they already possess; this is certainly true for the root of the broadcast. But since there are pieces that must travel $p-1$ hops to arrive at all of the nodes, we keep the algorithm symmetric by having all pieces circulate to all nodes during this phase. The resulting formula for $p = 2^d$ and $n$ a multiple of $p$ is:

$$\begin{aligned} T_{\mathrm{sb1}} &= \textstyle\sum_{i=0}^{d-1}[\alpha + \frac{n}{2^{i+1}}\,\beta] + \sum_{i=1}^{p-1}[\alpha + \frac{n}{2^d}\,\beta] \\ &= (d+p-1)\,\alpha + 2\,\frac{p-1}{p}\,n\,\beta \\ &= (p+\log_2(p)-1)\,\alpha + 2\,\frac{p-1}{p}\,n\,\beta \end{aligned} \tag{9}$$

(The formula for general $p$ and $n$ is more complicated. We present the simplified version for clarity.) Compared to RSbcast, an extra $p-1$ startups are incurred, but the coefficient of $n\,\beta$ in the transfer time has been reduced from $\log_2(p)$ to $2\,\frac{p-1}{p}$.

## 5.2 2-D scatter-collect

For a two dimensional mesh, the 1-D version of scatter-collect can be used, but the ring of nodes that passes the buckets around during the collect phase has a length of $p-1$, which can be shortened by performing the algorithm in each dimension separately:

1. scatter in columns: Perform the scatter in the root node's column. At the end of this phase, the original vector is split into $r$ pieces distributed among the nodes in the column.

2. scatter in rows: Each row performs a scatter independently. Each node in the root node's column is the root for its row. At the end of this phase, the piece that "belongs" to this row is split into $c$ pieces distributed among the nodes in the row. Over the whole mesh, the original vector has been split into $rc = p$ pieces and distributed across the mesh.

3. buckets in rows: Each row independently forms a logical ring and circulates the pieces until every node possesses the entire piece belonging to that row.

4. buckets in columns: At this point, each column independently forms a logical ring, and circulates the pieces until every node possesses the entire vector.

We model the time for the algorithm as:

$$\begin{aligned} T_{\mathrm{sb2}} &= \textstyle\sum_{i=0}^{d_1-1}[\alpha + \frac{n}{2^{i+1}}\,\beta] + \sum_{i=0}^{d_2-1}[\alpha + \frac{n}{2^{d_1}}\,\frac{1}{2^{i+1}}\,\beta] \\ &\quad \textstyle + \sum_{i=1}^{c-1}[\alpha + \frac{n}{2^{d_1}\,2^{d_2}}\,\beta] + \sum_{i=1}^{r-1}[\alpha + \frac{n}{2^{d_1}}\,\beta] \\ &= (d_1+d_2+r+c-2)\,\alpha + 2\,\frac{p-1}{p}\,n\,\beta \\ &= (\log_2(p)+r+c-2)\,\alpha + 2\,\frac{p-1}{p}\,n\,\beta \end{aligned} \tag{10}$$

(Again, we have simplified the equation by assuming that $p$ is a power of two and $n$ is an integer multiple of $p$.) Instead of $p-1$ extra startups (compared to RSbcast), there are $r+c-2$ extra startups. When $r = c$ (i.e., the mesh is a square), this becomes $2\sqrt{p}-2$.

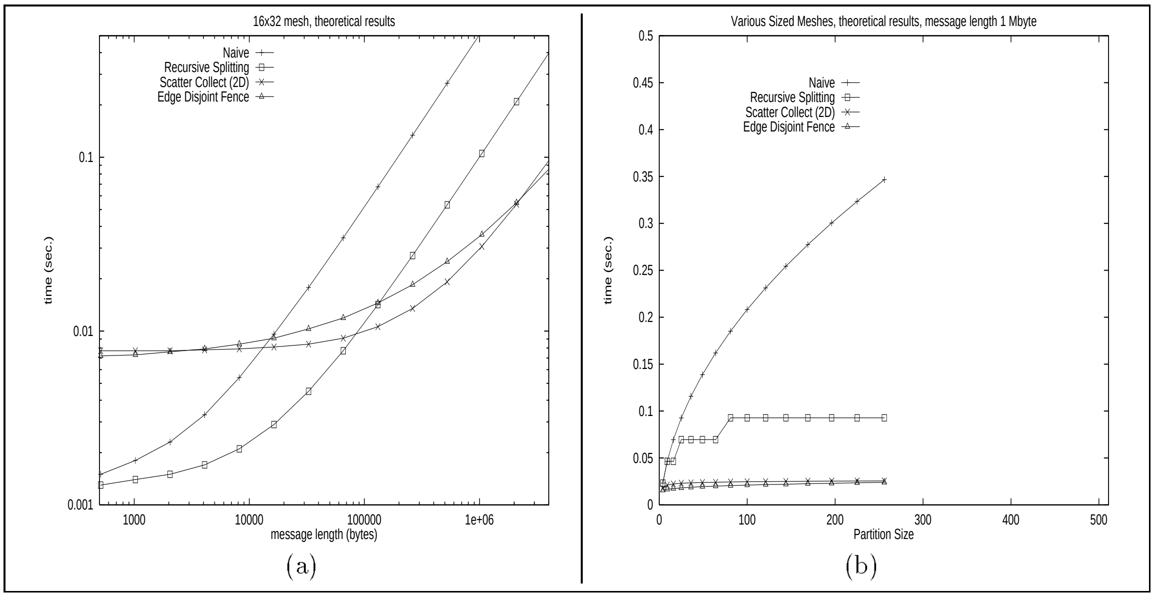



Figure 7: Predicted time for the various algorithms on an idealized machine: performance as a function of message length (a) and of grid size (b). The grid sizes were chosen to equal $i \times j$, where $i = 2, \ldots, 16$ and $j = i, 2i$.

# 6 Comparison of Algorithms

In this section, we compare the performance of the different algorithms. First, we examine the performance under the idealized model given in Section 2. Next, we adjust the model to more closely fit the Intel Paragon, the architecture available to us that resembles our model. Finally, we compare the predicted execution times under this new model with the times observed on the Paragon.

## 6.1 Theoretical Comparison

In Figure 7, we report the predicted execution times of the algorithms on an idealized architecture that satisfies Assumptions 1-8 in Section 2. The machine constants are fixed to correspond approximately to those of the Paragon, with $\alpha = 70\mu$sec and $\beta = .011\mu$sec. Equations 1, 6, 8 and 10 were used for the different curves.

On such an idealized architecture, all but the naive algorithm show merit for a region of vector lengths.

## 6.2 Adjustments Necessary to Model the Paragon

The closest architecture available to us to check our theoretical results is the Intel Paragon. The Paragon routing scheme uses the x-direction first, then the y-direction, as in Lemma 2. It conforms to all of the

Assumptions 1–8 *except Assumptions 4* and 7. The interconnection network on the Paragon is bidirectional. However, in effect, the Paragon can only send or receive one message at a time. If a node sends and receives simultaneously, the effective $\beta$ is essentially doubled. We will use $\beta$ for the former, $\beta_2$ for the latter. In addition, there is excess bandwidth in the network, so that all messages traversing a given link timeshare a bandwidth of $\beta_{net}^{-1}$. We adjust the models for the different algorithms accordingly:

**Naive MST broadcast:** Assuming $\beta_{net} = \beta$, for the naive minimum spanning tree broadcast the estimated time is still given by $T_{\text{naive}}$ (Eqn. (1)). This is somewhat pessimistic, since on the Paragon $\beta_{net} < \beta$.

**Recursive Splitting broadcast:** The estimated time is still given by $T_{\text{rs}}$ (Eqn. (6)).

**Scatter/Collect broadcast:** During the Scatter, all nodes are only receiving or sending at a given time, not both. However, during the collect all nodes send and receive simultaneously. As a result, the estimated time becomes:

$$\lceil \log(p) \rceil \alpha + \frac{p-1}{p} n\beta + (r+c-2)\alpha + \frac{p-1}{p} n\beta_2$$

**Edge Disjoint Fence broadcast:** At each step in this algorithm, most nodes either send, or send and receive simultaneously. Due to the timing of the messages that wrap around, we have observed on a network simulator that for some nodes, two messages arrive in the same step. Moreover, the simulator also shows that the resulting interference creates "bubbles" in the process, leading to further degradation of performance. As a result, a better estimate of the time for transferring an item (byte) is a number larger than $\beta_2$, we will call it $\beta_3$, leading to a predicted time of:

$$T_{\text{edf}} = (k_{\text{opt}} + r + c)(\alpha + \frac{n}{k_{\text{opt}}}\beta_3) \qquad (11)$$

with:

$$k_{\text{opt}} = \begin{cases} 1 & \text{if } \sqrt{(r+c)n\beta_3/\alpha} < 1 \\ n & \text{if } \sqrt{(r+c)n\beta_3/\alpha} > n \\ \sqrt{(r+c)n\beta_3/\alpha} & \text{otherwise} \end{cases}$$

When the number of packets is small enough that the wrapping doesn't interfere with subsequent messages, $\beta_3$ in these equations should be replaced by $\beta_2$. In our estimates we simply use Equation 11.

Our implementations used forced messages, which means that the receiver is assumed to be ready for the message when it arrives. This doubles the bandwidth between nodes, but also doubles the latency, since a synchronization message must be sent. As a result, $\alpha = 140\mu$sec. $\beta = .011\mu$sec., $\beta_2 = .021\mu$sec., $\beta_3 = .042\mu$sec., for OSF release R1.2.

In Figures 8, 9 and 10, we report the observed versus predicted times for the various algorithms, for a broadcast rooted at node 0. The predicted and observed timings agree enough to claim that the models are useful. In Figures 8, 10, and 11 (b) we report the observed time for the EDF algorithm when the theoretical optimal $k_{\text{opt}}$ is used. In Figure 11 (b), we report the observed time as a function of the root node of the broadcast. The naive broadcast is very dependent on the root due to network conflicts, but the other algorithms are not noticeably affected.

Some interesting observations can be made about the data:

- In reality it becomes extremely difficult to model a parallel architecture like the Paragon. Depending on the very specific nature of communication, bandwidth and latency changes. Indeed, even when vector lengths communicated change, the observed bandwidth and latency changes.
- The odd data points for the EDF algorithm in Fig. 10 are due to the erratic behavior of the EDF algorithm. Interestingly enough, broadcasting a fixed length message on a small number of nodes takes longer that on a large number of nodes!
- In practice the scatter/collect outperforms the theoretically better EDF algorithm.

# 7 Related Work

As mentioned previously, the state of the art in broadcasting on hypercubes is [5, 6]. Our approach to edge-disjoint fences is closely related to the work in [4], where the embedding of edge-disjoint trees in wraparound (torii) meshes is discussed. The true wraparound links provide a mesh that has roughly half the diameter of the worm-hole meshes we consider. Their trees are much more complicated than the ones presented here; it is not clear whether their construction would lead to network conflict in a worm-hole mesh. In [11], a broadcast is presented that has somewhat of a flavor of our "scatter-collect". In essence, the author followed our suggestion that the broadcast can be implemented as a modified global summation and used some of the techniques for such algorithms developed in [1, 2, 3, 12]. The resulting algorithms are not asymptotically optimal, but do avoid network conflicts. They are limited to meshes that contain a power-of-two number of nodes, with extensions for general meshes that double the cost of the algorithms.

# 8 Other applications of the techniques

Global combine operations, which leave a result on a single node, require communication that is the inverse of the broadcast. Such a combine is referred to as a "fan-in"; a broadcast is thus a "fan-out". All techniques presented in this paper can be extended to this communication operation. As mentioned in the introduction, the theory developed for the MST broadcast can also be applied to the scatter and gather operation. Notice that for the combine-to-one operation and the gather the communication must be performed in the opposite direction. As a result, the minimum spanning tree must be adjusted so that messages still satisfy the

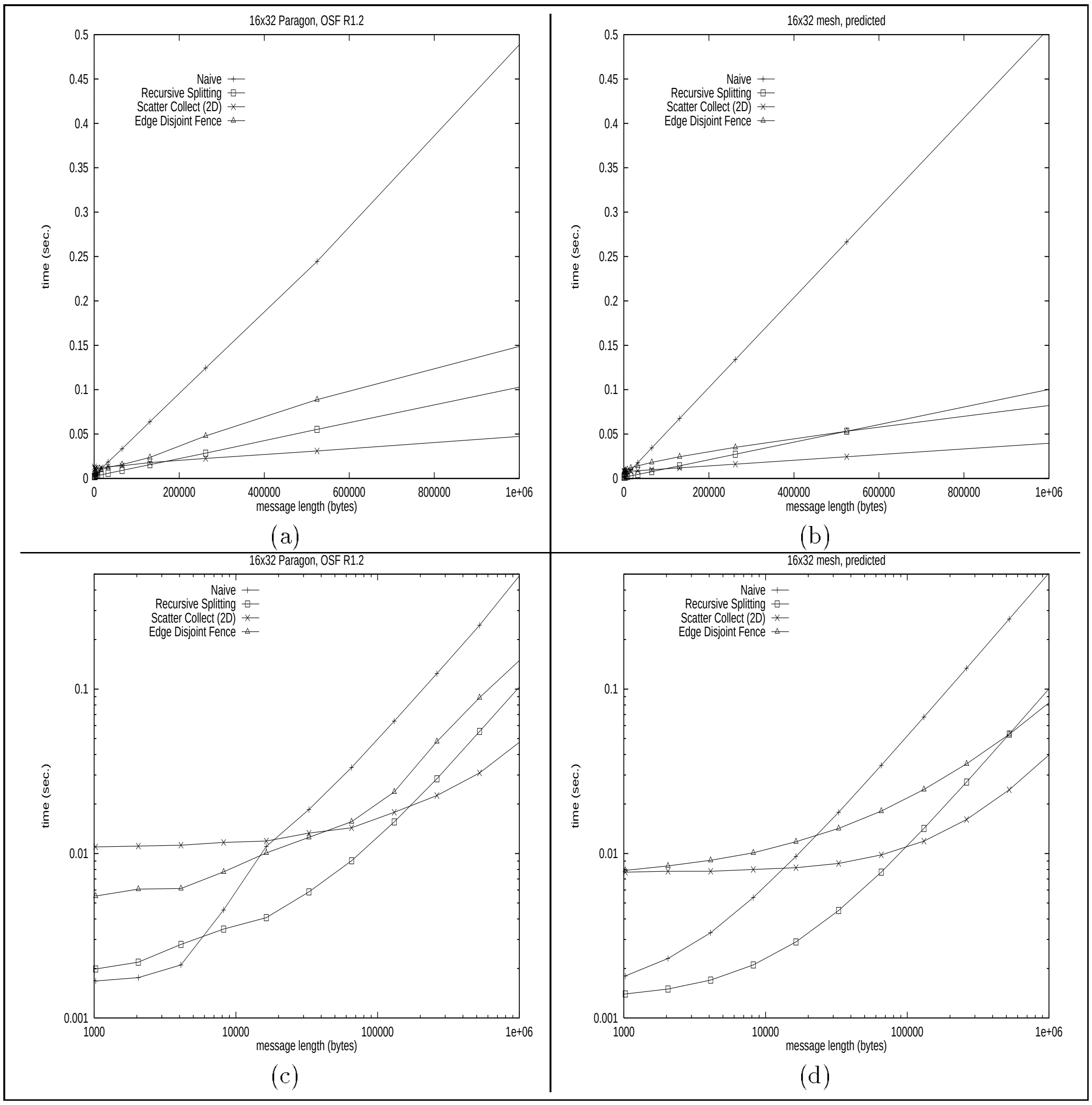



Figure 8: Observed time vs. predicted time for the Paragon, as a function of vector length.

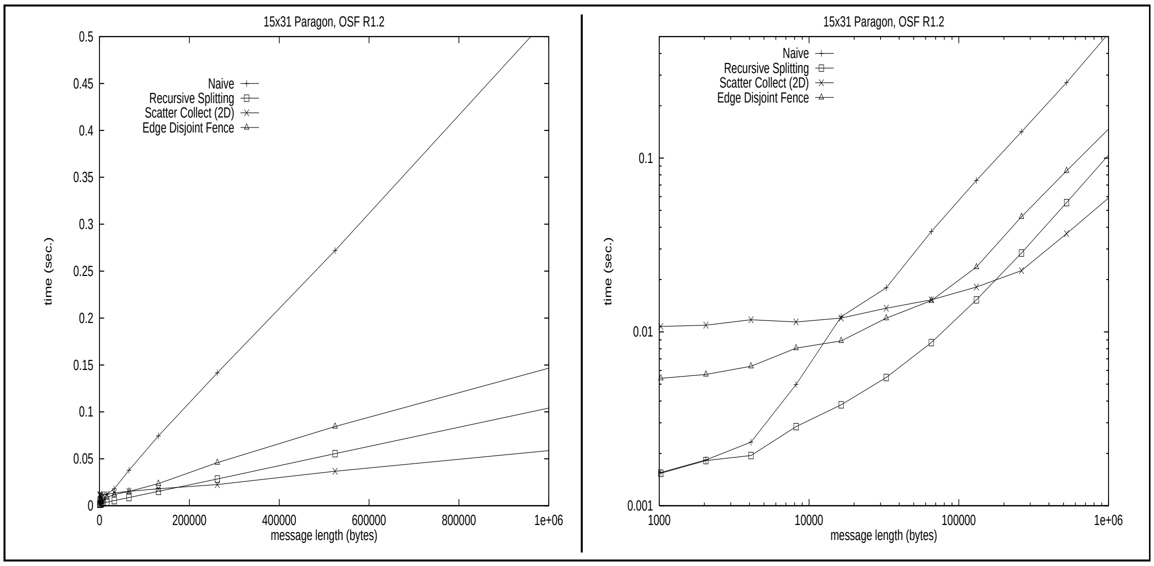



Figure 9: Observed time time for an odd-sized partition of the Paragon, as a function of vector length. Notice the observed behavior is very similar to that of the slightly larger, power-two partition reported in the previous graphs.

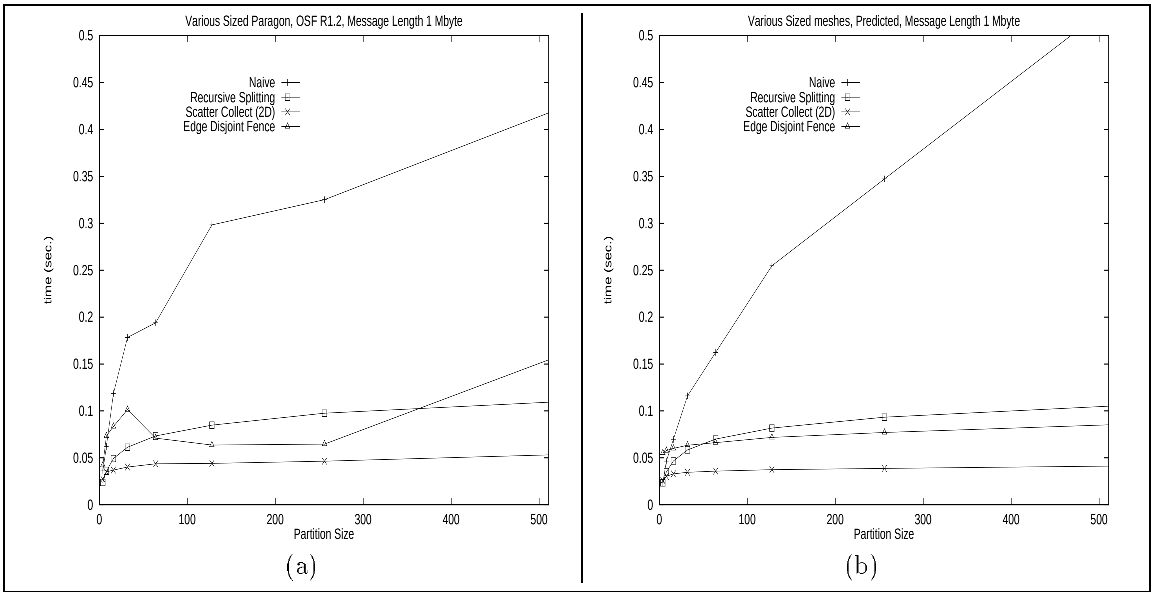



Figure 10: Observed time vs. predicted time for the Paragon, as a function of mesh size.

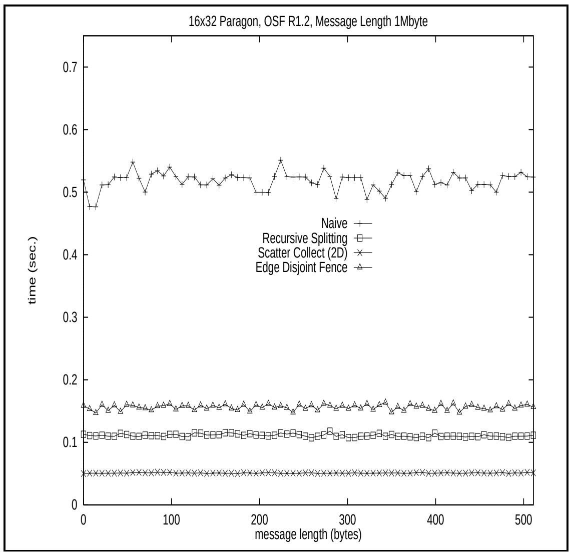



Figure 11: Time as a function of the root node of the broadcast.

conditions of the Lemmas in Section 3. We leave it as an exercise to the reader to design the appropriate implementations.

# 9 Conclusions

Our work makes clear that efficient broadcast algorithms are possible for mesh architectures. Their non-recursive nature—compared to hypercubes—does require more careful analysis in order to arrive at efficient implementations. While the idealized model provides insight, a more detailed model is also presented to more closely fit the specific architecture on which we performed our experiments.

The conclusions that we can draw from this work are the following:

- For short vectors, broadcasting on a mesh is as efficient as on a hypercube.
- Asymptotically, for long vectors, in theory one can broadcast in essentially the same time on a mesh as on a hypercube.
- In practice, we can conclude that as a general approach, this kind of pipelining is extremely architecture dependent, its performance is very erratic and unpredictable (see Figure 10), and it is an extremely difficult algorithm to implement efficiently.
- For long vectors, the scatter-collect algorithm has much nicer properties:

- It is within a factor two of optimal.
- It is very predictable.
- The details of how the scatter and collect are implemented is architecture specific, but not the general approach. Any scatter and collect that does not incur network conflicts will suffice, at the potential expense of additional latency overhead.

Ultimately, hybrid algorithms that combine the algorithm that is best for short vectors with an efficient algorithm for long vectors will need to be developed. We are currently investigating such hybrids.

## Acknowledgements

This research was performed in part using the Intel Paragon System operated by the California Institute of Technology on behalf of the Concurrent Supercomputing Consortium. Access to this facility was provided by Intel Supercomputer Systems Division and the California Institute of Technology.

## References

[1] M. Barnett, D. Payne, and R. van de Geijn. Optimal broadcasting in mesh-connected architectures. Technical Report TR-91-38, Department of Computer Sciences, The University of Texas at Austin, Dec. 1991.

[2] M. Barnett, R. Littlefield, D.G. Payne, and R. van de Geijn. Efficient Communication Primitives on Mesh Architectures with Hardware Routing. *Sixth SIAM Conf. on Par. Proc. for Sci. Comp.*, Norfolk, Virginia, March 22-24, 1993.

[3] M. Barnett, R. Littlefield, D.G. Payne, and R. van de Geijn, Global Combine on Mesh Architectures with Wormhole Routing, *7th International Parallel Processing Symposium*, pages 156–162, IEEE Computer Society Press, Newport Beach, CA, April 13-16, 1993.

[4] J.-C. Bermond, P. Michallon, and D. Trystram. Broadcasting in wraparound meshes with parallel monodirectional links. *Parallel Computing*, 18:639–648, 1992.

[5] C.-T. Ho and S. L. Johnsson, Distributed Routing Algorithms for Broadcasting and Personalized Communication in Hypercubes. *Proceedings of the 1986 International Conference on Parallel Processing*, pages 640–648, IEEE, 1986.

[6] C.-T. Ho and M.T. Raghunath. Efficient communication primitives on hypercubes. Technical Report RJ 7932 (72915), IBM, Jan. 1991.

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

[8] R.J. Littlefield, "Modeling Node Bandwidth Limits and Their Effect on Vector Combining Algorithms" Pacific Northwest Laboratory, no. PNL-SA-20425, 1992.

[9] L. M. Ni and P. K. McKinley. A survey of wormhole routing techniques in direct networks. *IEEE Computer*, 26(2):62–76, Feb. 1993.

[10] Y. Saad and M. H. Schultz. Data Communiciation in Parallel Architectures. Yale University Research Report YALEU/DCS/RR-461, 857–873, 1986.

[11] S. R. Seidel. Broadcasting on Linear Arrays and Meshes. Oak Ridge National Laboratory Technical Report ORNL/TM-12356, Mar. 1993.

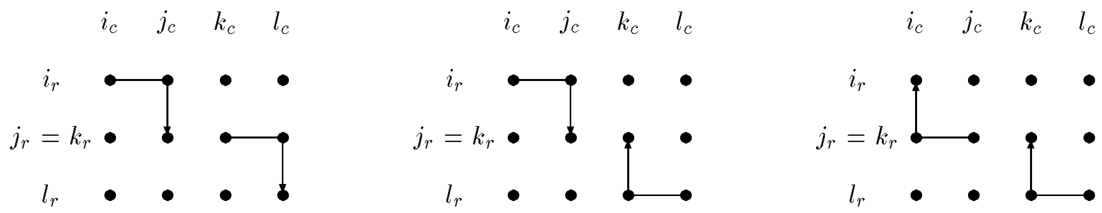
[12] R. A. van de Geijn. Efficient Global Combine Operations. In *Sixth Distributed Memory Computing Conference Proceedings*, pages 291–294. IEEE Computer Society Press, 1991.

# A Proof of Lemma 2

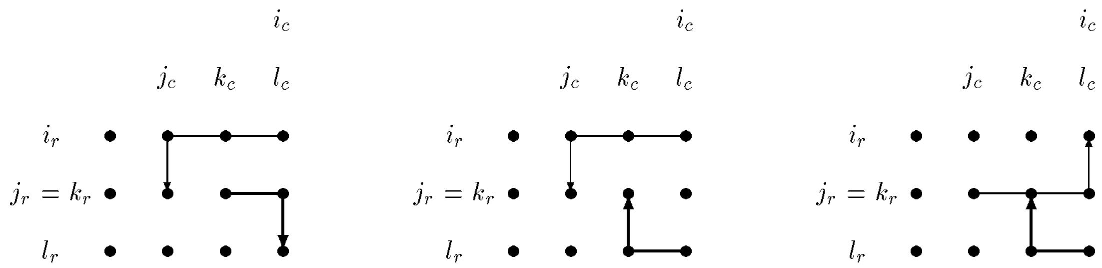
Assume $i < j < k < l$ and let $(x_r, x_c)$ denote the row and column indices of node $x$ ($x \in \{i, j, k, l\}$). Row-major ordering means that $i < j \equiv ((i_r = j_r \wedge i_c < j_c) \vee i_r < j_r)$. Therefore $i < j < k < l$ implies $i_r \leq j_r \leq k_r \leq l_r$. If $j_r < k_r$, then the messages will be routed in disjoint sets of links, due to the minimum length path property (each pair is in a separate rectangular sub-mesh). Without loss of generality, assume $j_r = k_r$, which implies $j_c < k_c$. In the proof, the left column is for the pair of paths $\langle i, j \rangle$ and $\langle k, l \rangle$, the middle column is for the pair $\langle i, j \rangle$ and $\langle l, k \rangle$, and the right column is for the pair $\langle j, i \rangle$ and $\langle l, k \rangle$. Note that the pictures compress certain cases. For instance the first row of pictures show $i_r < j_r$ and $k_r < l_r$, but they are also valid for $i_r = j_r$ or $k_r = l_r$.

Assume x-direction first routing.

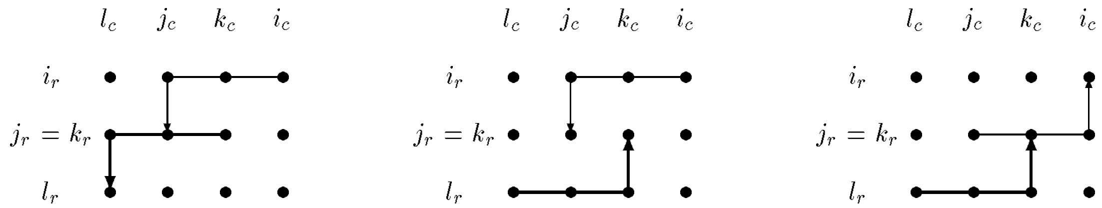
- $i_c \leq j_c$ and $k_c \leq l_c$




- $i_c > j_c$ and $k_c \leq l_c$. **Note:** $i_c > j_c \Rightarrow i_r < j_r$.




- $i_c > j_c$ and $k_c > l_c$.



- $i_c \leq j_c$ and $k_c > l_c$.