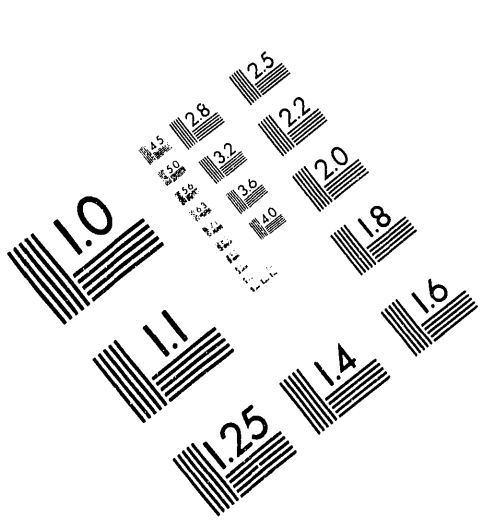
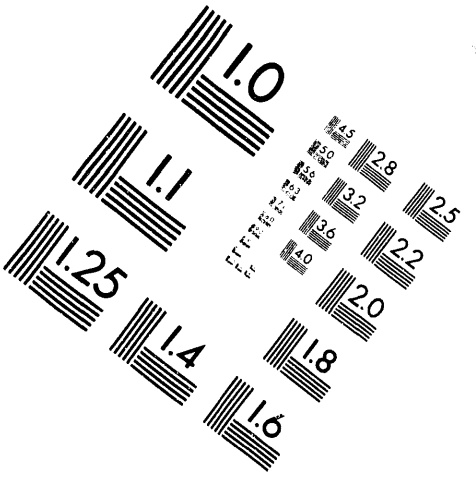
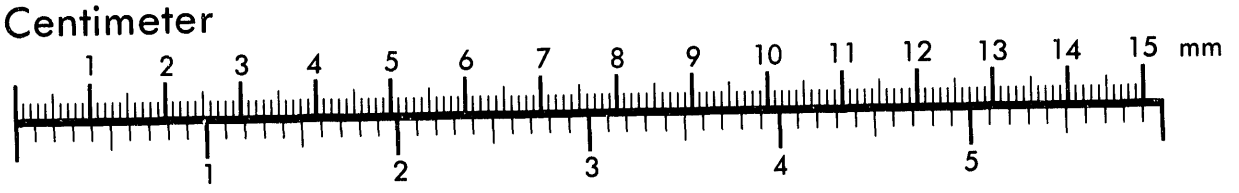
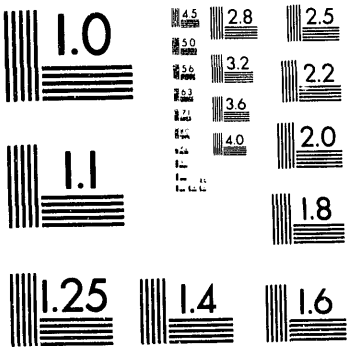
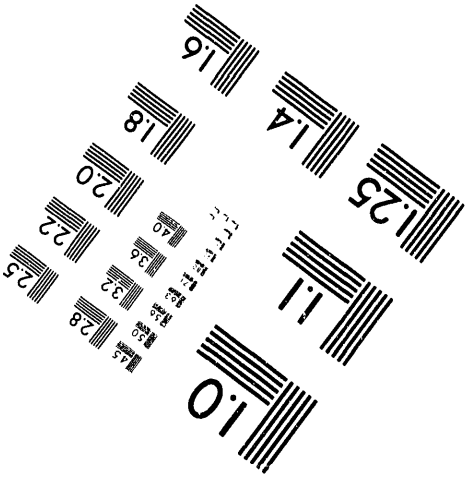


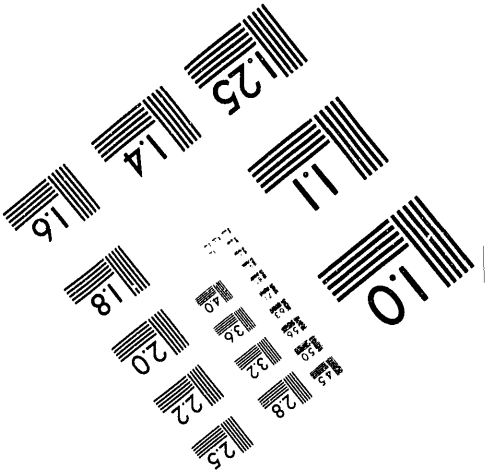
Association for Information and Image Management
1100 Wayne Avenue, Suite 1100
Silver Spring, Maryland 20910
301/587-8202











1 of 1

ORNL/TM-12356

Engineering Physics and Mathematics Division

Mathematical Sciences Section

BROADCASTING ON LINEAR ARRAYS AND MESHES

Steven R. Seidel

Department of Computer Science
Michigan Technological University
1400 Townsend Drive
Houghton, Michigan 49931-1295

steve@cs.mtu.edu

Date Published: March 1993

Research was supported by the Applied Mathematical Sciences Research Program and the Atmospheric and Climate Research Division of the Office of Energy Research, U.S. Department of Energy, and by NASA Ames Research Center grant NAG2-757.

Prepared by the
Oak Ridge National Laboratory
Oak Ridge, Tennessee 37831
managed by
Martin Marietta Energy Systems, Inc.
for the
U.S. DEPARTMENT OF ENERGY
under Contract No. DE-AC05-84OR21400

Contents

BROADCASTING ON LINEAR ARRAYS AND MESHES

Steven R. Seidel

Abstract

The well known spanning binomial tree broadcast algorithm is generalized to obtain several new broadcast algorithms for linear arrays and meshes. These generalizations take advantage of bidirectional communication, the connectivity of two-dimensional meshes, and the difference between node-to-network and network-to-network bandwidth. It is shown how these algorithms can be further generalized so that any node can be the source of the broadcast message. A partitioning scheme is given that allows these algorithms to be used on linear arrays and meshes of any size. One of these algorithms, the bidirectional spanning tree broadcast, always has lower cost than the recursive halving broadcast for linear arrays. All of these algorithms offer significant performance improvements over the basic spanning tree broadcast. These algorithms do not rely on a knowledge of machine-dependent constants for network bandwidth and latency, so their performance is not as sensitive to changes in machine characteristics as that of hybrid and pipelined algorithms. Performance measurements are given for some of these broadcast algorithms on the Intel Delta mesh.

1. Introduction

A *broadcast* is an operation where one processor of a multicomputer has a message that must be copied to each of a set of processors. Solutions to this problem for various MIMD architectures have been widely studied. Much attention has been given to this problem for linear arrays and meshes [2,3,8,14,15] because of the recent availability of such machines. One group of broadcast algorithms [3,14,15] transmits the message pipeline fashion, in packets. While these algorithms have good theoretical performance, specific constants representing network bandwidth and latency are used at execution time to compute optimal packet lengths. This makes these algorithms sensitive to changes in the system on which they are implemented. In addition, such algorithms can be more difficult to implement on a mesh than on a hypercube because a mesh lacks the vertex and edge symmetry that is so useful for the implementation of algorithms on hypercubes. So far there have been no reports on the performance of these algorithms on existing meshes. Another approach to algorithm design is to construct hybrid algorithms. Examples of hybrid algorithms for other communication problems are given in [5,9], but such algorithms also use a knowledge of network constants to determine which strategy to apply. Neither pipelined nor hybrid algorithms are considered here.

The work presented here offers several new solutions to the broadcast problem for linear arrays and meshes based on the familiar spanning binomial tree (the communication pattern used in "recursive doubling" on hypercubes) [12] and on dimensional exchanges (also known as the "butterfly"). These algorithms are extensions of those given in [2]. They will perform well even as machine characteristics change because their design avoids execution-time dependence on constants that represent network performance characteristics. In particular, they do not use constants representing network bandwidth and latency to determine packet sizes or to decide among strategies, as is done in pipelined and hybrid algorithms. The recursive halving broadcast algorithm is also considered here because it shares this independence. This broadcast algorithm is only a slight modification of the recursive halving global combine algorithm given by van de Geijn in [1,16].

The next section describes the communication model on which the analysis of communication algorithms will be based. The broadcast algorithms and their costs are given in Section 3. In Section 4 it is shown how these algorithms can be generalized so that any node is the source of the broadcast message. Generalizations of these algorithms to linear arrays and meshes of any size are given in Section 5. Section 6 presents the predicted and observed performance of some these algorithms on the Intel Delta mesh.

2. Communication Model

This work considers solutions to the broadcast problem on linear arrays of n processors and on 2-dimensional meshes of $n = n_1 \times n_2$ processors. The nodes of a linear array are numbered from 0 through $n-1$. The nodes of a mesh are numbered from $(0,0)$ through (n_1-1, n_2-1). Each node is connected by a pair of communication links (one in each direction) to each of its immediate neighbor(s) in the horizontal and vertical directions. There are no wrap-around links. Each node can concurrently transmit one message and receive one message. Circuit switched message passing with wormhole routing is used. The route taken by a message in a linear array is just the shortest path between the sending and receiving nodes. A message in a mesh is routed horizontally until it reaches the column containing the receiving node and then it is routed vertically to the receiving node. Messages can be routed through a node without affecting its performance as a sender or receiver. Finally, it is assumed that each receiving node allocates a buffer for each incoming message before that message arrives. Under this assumption the sending node can transmit a message without prior handshaking with the receiver. (This is the "forced" message passing protocol [11].)

A simple communication model describes the cost of sending a message of m bytes as $am + b$, where b is the latency and a is the per-byte transmission cost. A close examination of real message passing networks reveals a much larger collection of factors that can affect the cost of communication algorithms. Some of these factors are:

- contention for communication ports and links,
- choice of message passing protocol,
- packet permutation costs (message packet copying) within a node,
- bandwidth differences between different parts of the message passing network, such as from the node to the network "gateway", and on the network itself (between "gateways"),
- the length of the circuit over which the message travels,
- the effects of message packetization performed by the node operating system,
- costs of arithmetic or logical operations on the message, such as in the combine operation,
- the distinction between the cost of a send operation measured in the sending node alone, and the cost measured from the initiation of a send operation to the completion of the corresponding receive operation,
- synchronization costs,
- communication algorithm execution overhead (loop control, *etc.*),

- delays contributed by operating system interrupt processing,
- delays contributed by concurrent computation,
- the effect of message traffic on the network that is not under the control of the application programmer, such as that caused by I/O operations and other message traffic generated by the operating system, and
- the effect of caches, FIFOs, and other hardware features.

Along with transmission cost a and latency b, the first four factors (link and port contention, protocol choice, permutation costs, and bandwidth differences) significantly affect the cost of global communication on the Intel Delta mesh, so these factors are included in the communication model presented here. The remaining factors, many of which have relatively small affects on cost, will not be considered here.

The communication model of the Delta that will be used here is the same as that presented in [1], with the addition of a term for permutation costs. (See [11] for a general description of the message passing network of that machine.) This model distinguishes two kinds of transmission costs: a is the per-byte cost of moving data from a processor onto the network and $\bar{a}$ is the cost of moving data over the network itself. This means that there is a path from (or to) a node to a network "gateway" and that there is a circuit connecting the "gateways" of the source and destination nodes. These two costs are not cumulative, they simply represent the capacity of these two components of the network. It is assumed that $a = 2^{\nu}\bar{a}$ for some integer $\nu \geq 0$.

Link contention occurs when the paths taken by two or more messages have one or more links in common. As long as no more than 2^{ν} messages share any one link at a given time, link contention does not add to cost of sending a message. *Port contention* occurs when one or more messages arrive at a node at the same time. Those messages can arrive over distinct incoming links or, if $\nu > 0$, they can arrive on the same link. In both cases port contention adds to cost of sending a message.

In the absence of communication port contention, the cost of transmitting a message of m bytes is expressed as $\lceil k/2^{\nu} \rceil am + b$, where k is the largest number of message circuits that share a link during the transmission of the message. Suppose that $k \leq 2^{\nu}$ send operations are initiated by k distinct nodes. If all of the receiving nodes are distinct, that is, there is no port contention, then the latency of each send operation is overlapped and the total cost of these send operations is just $am + b$, regardless of whether or not any of the message circuits have one or more links in common. However, when $k = 2^i$ for some $i > \nu$, the transmission cost is $2^i\bar{a}m + b = 2^{i-\nu}am + b$.

Finally, certain communication algorithms, such as the complete exchange algorithms given in [5] and [6], require significant amounts of message packet movement within individual nodes.

Such internal data movement is also required in the recursive halving broadcast algorithms described in the next section. During execution of these algorithms the next message to be sent is formed by permuting the message packets that have already arrived. In these algorithms each node moves as much data internally as it does over the network. A constant ρ that represents the cost of moving one byte of data from one location to another within a node is thus included in the cost analysis of the algorithms given in the next section. Also note that permuting message packets usually requires that additional storage be provided for message packets. This typically amounts to a doubling of storage requirements, which, when long messages are involved, can be a significant factor in the choice of algorithms.

3. Broadcast Algorithms

Several broadcast algorithms for linear arrays and meshes are considered. Each algorithm is based on communication patterns commonly used on hypercubes, such as spanning binomial trees and dimensional exchanges. In this section it is assumed that a linear array consists of $n = 2^d$ nodes and that a mesh consists of $n = 2^{d_1} \times 2^{d_2}$ nodes. It is also assumed that node 0 contains a message of length m to be broadcast and that all other nodes are blocked, waiting for node 0 to begin the broadcast operation. The *cost* of a broadcast operation is measured from the time node 0 begins the broadcast to the time the last node receives the message.

The familiar spanning tree broadcast algorithm is considered first. Several improvements of that basic algorithm are given. These improvements take advantage of:

1. the additional bandwidth offered by $\nu > 0$,
2. the additional bandwidth offered by bidirectional links, and
3. the increased connectivity of meshes over linear arrays.

For comparison, the recursive halving broadcast algorithm is also described at the end of this section.

3.1. Spanning tree broadcast on a linear array

3.1.1. ST broadcast, $\nu = 0$

The first algorithm is based on the familiar spanning binomial tree that is used in the recursive doubling broadcast algorithm for hypercubes [12]. This algorithm will be called the *spanning tree (ST) broadcast* and was described earlier in [2] for linear arrays and meshes. A ST broadcast on a linear array of 2^d nodes takes d steps. On the i^{th} step ($1 \leq i \leq d$), each node j that already has a copy of the message sends it to node $j \oplus 2^{d-i}$, where $\oplus$ denotes bit-wise exclusive OR.

This is illustrated for a linear array in Figure 1(a). The corresponding spanning tree is shown in Figure 1(b). Each arc of the tree is labeled with the step at which it carries the message.

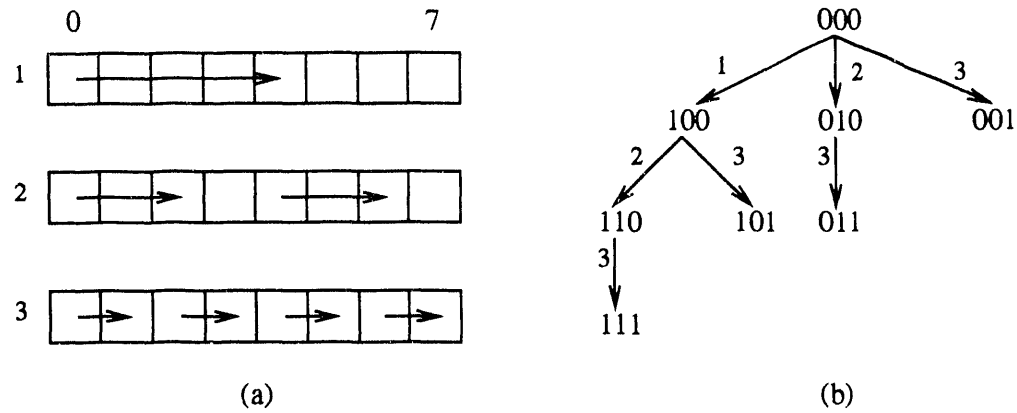


Figure 1: Spanning tree broadcast on a linear array.

The spanning binomial tree on which the ST broadcast algorithm is based is also used in all of the other broadcast algorithms considered here. The nodes of such trees are numbered in binary and the bits of each node number are indexed from most to least significant by $d-1, d-2, ..., 0$. (From a purely graph-theoretic point of view, these trees are not spanning trees of linear arrays or meshes. However, these trees are useful for describing the scheduling and routing of messages in those networks and they will continue to be referred to as spanning trees here.) For purposes here, a *spanning tree with root 0* is a directed graph of $n = 2^d$ nodes in which each node i has children whose node numbers are obtained by complementing exactly one of the trailing zeros (if any) of i. To determine the node numbering of a spanning tree with root other than 0, exclusive OR the node number of each node in the tree with the node number of the new root. For more details about the properties of these trees see [12].

In the spanning tree of a linear array of 2^d nodes, some tree arcs represent circuits of several links in the linear array and some links in the linear array are used in several tree arcs. Since some of the tree arcs carry messages simultaneously during this broadcast algorithm, there might be link contention. (This possibility does not arise in a hypercube because there is a one-to-one correspondence between tree arcs and hypercube links.) Even though some arcs in the tree share the same links, at each step only disjoint sets of links are used. It was shown in [2] that the ST broadcast algorithm causes no port or link contention if node 0 is the root of the spanning tree. Also, there are no packet permutation costs in this algorithm because the message is not divided into packets. Thus, the cost of the ST broadcast algorithm is

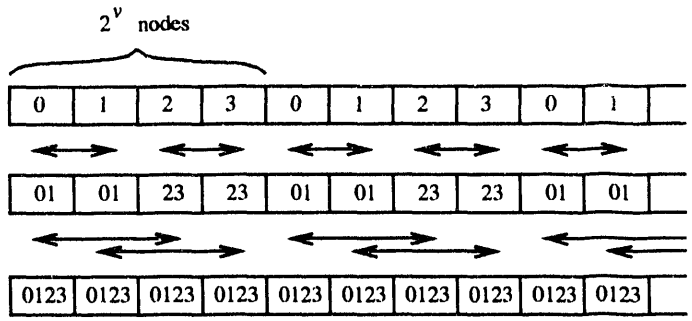
$$d(ma + b) \quad \text{for a linear array of } 2^d \text{ nodes.} \tag{1}$$

There are two other communication problems whose solutions are used frequently in the broadcast algorithms given here. Solutions to these two problems, based on spanning trees, are now described. Many of the algorithms that follow make use of the *distribute* operation in which one node sends a distinct message to each other node in the network. This operation is also called a *scatter* or a *one-to-all personalized communication* [12]. In most of the broadcast algorithms that follow, the messages to be distributed arise by partitioning the message that is to be broadcast. In a linear array or mesh of $n = 2^d$ nodes, each of the distributed messages, called *packets*, has length m/n, where m is the length of the message to be broadcast. The distribute algorithm that is used here is based on a spanning tree. A message of length m is distributed to all the nodes in the tree by halving it at each step until each node has received its packet. The cost of this distribute algorithm is

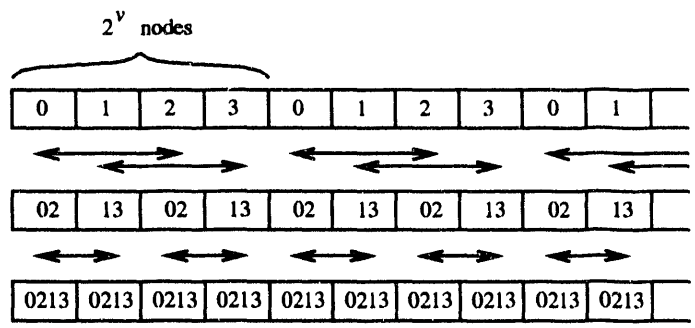
$$\sum_{i=1}^{d}(\frac{ma}{2^i} + b) = (1 - \frac{1}{2^d})ma + db.$$

The other problem of interest is called the *all-to-all broadcast*. In this problem, *each* node has a message (typically, a packet of length m/n) that must be broadcast to all other nodes. This problem is easily solved by exchanging packets between nodes whose node numbers (written in binary) differ by one bit. On hypercubes this algorithm is known as a *dimensional exchange* and the same term will be used here. Note that the message length doubles at each step of this algorithm and so its cost is the same as the cost of the distribute algorithm described above, provided there is no link contention. However, on linear arrays and meshes a dimensional exchange can give rise to both link contention and permutation costs. The issue of link contention is considered in detail in Sections 3.5 and 3.6, where the dimensional exchange is a critical part of the recursive halving broadcast algorithm. Until then dimensional exchanges will be done only on subarrays of 2^ν nodes and on submeshes of $2^\nu \times 2^\nu$ nodes. Link contention does not arise in arrays of these sizes.

Permutation costs can arise during a dimensional exchange if there is some inherent ordering among the packets that are exchanged. There is always such an ordering associated with the packets exchanged in the broadcast algorithms studied here. For example, it will often be the case that each contiguous subarray of 2^ν nodes contains distinct packets numbered $0, 1, ..., 2^{\nu-1}$, as shown in the two examples in Figure 2. It will be required that each subarray does an all-to-all broadcast of the packets within that subarray so that each node receives a copy of each packet. In addition, the packets must ultimately be ordered by their index. Figure 2(a) illustrates that the ordering of packets is preserved by exchanging with nearest neighbors first. This corresponds to selecting destinations by complementing the sender's node number bits from least to most significant. The opposite ordering of destinations results in an out-of-order concatenation of



(a) right



(b) wrong

Figure 2: Dimensional exchanges of 2^ν packets among 2^ν nodes, $\nu = 2$.

packets, as shown in Figure 2(b). In this case an internal permutation of packets is required to achieve the desired packet ordering. Until the recursive halving algorithms are introduced in Section 3.5, all of the broadcast algorithms given here perform dimensional exchanges so that permutation costs are avoided. (Note that since the size of each subarray is limited to 2^ν, the overlapped pairs of exchanges do no exceed the capacity of the links, so there is no link contention during these exchanges.)

With the basic tools introduced above, we can now continue with the construction and analysis of broadcast algorithms.

3.1.2. ST broadcast on a linear array, $\nu > 0$

If $\nu > 0$ and if it is assumed that ν is an integer, the ST broadcast can be generalized to take better advantage of the available network bandwidth. Under these circumstances the linear array can be viewed as 2^ν interleaved subarrays each with $2^{d-\nu}$ nodes. The i^{th} of these subarrays, for $0 \leq i < 2^\nu$, consists of nodes numbered $j2^\nu + i$, for $0 \leq j < 2^{d-\nu}$. Figure 3 shows the two interleaved arrays as white and gray cells in a linear array of 8 nodes for the case of $\nu = 1$.

To broadcast the message on these interleaved arrays, the message is first distributed among nodes 0, 1,..., $2^\nu - 1$. The cost of this distribution is

$$(1 - \frac{1}{2^\nu})ma + \nu b.$$

When $\nu = 1$ this distribution phase amounts to only one step, as shown in the first step in Figure 3. Each of the nodes 0, ..., 2^ν then acts as the source node of a ST broadcast of a message of length $m/2^\nu$ on a subarray of $2^{d-\nu}$ nodes, at a cost of

$$(d - \nu)(\frac{ma}{2^\nu} + b).$$

This is shown in the second and third steps in Figure 3. Since there is no link contention during a ST broadcast there is also no link contention when 2^ν ST broadcasts are performed concurrently because the maximum number of messages that contend for any link is 2^ν. There is no port contention because none of the spanning trees have any nodes in common. Also note that the ST broadcasts leave the packets ordered in each contiguous subarray of 2^ν nodes just as they were after they were first distributed.

The final phase of the algorithm consists of collecting the packets to reconstruct the original message. This is accomplished by a dimensional exchange on each contiguous subarray of 2^ν nodes. This is the final step in Figure 3. Recall that there is no link contention during this phase since all communication is localized among contiguous subarrays of 2^ν nodes. Also,

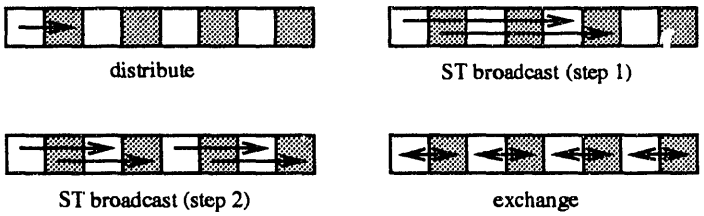


Figure 3: ST broadcast on a linear array, $\nu = 1$.

the dimensional exchanges can be done in an ordering that allows the original message to be reconstructed without the need for permuting the packets within each node. Thus, this phase has cost

$$(1 - \frac{1}{2^\nu})ma + \nu b.$$

The total cost is

$$(2 + \frac{d - \nu - 2}{2^\nu})ma + (d + \nu)b. \tag{2}$$

When $\nu = 0$ this cost reduces to $d(ma + b)$, as it should.

3.2. Spanning tree broadcast on a mesh

3.2.1. ST broadcast on a mesh, $\nu = 0$

A ST broadcast on a mesh, based on the algorithm for linear arrays, is shown in Figure 4. In this algorithm the message is first broadcast to the nodes in the leftmost column of the mesh, then each node in that column broadcasts the message to the nodes in its row. The cost of this algorithm is

$$(d_1 + d_2)(ma + b) \quad \text{for a mesh of } 2^{d_1} \times 2^{d_2} \text{ nodes,}$$

and it is easy to see that there is no link or port contention and that there are no permutation costs.

This algorithm does not take very good advantage of the connectivity of the mesh. Better advantage is taken by treating the mesh as four interleaved $2^{d_1-1} \times 2^{d_2-1}$ submeshes. This viewpoint is illustrated in the first two frames of Figure 5 where each of the four submeshes is given a unique hatching pattern. The message is first distributed to the 2×2 block of nodes in the upper left corner of the mesh so that each of the four nodes in that block has one quarter of the message. This is shown in the first two steps of Figure 5. The cost of these steps is

$$\frac{3}{4}ma + 2b.$$

Each of the four corner nodes then uses the ST broadcast algorithm described in the previous

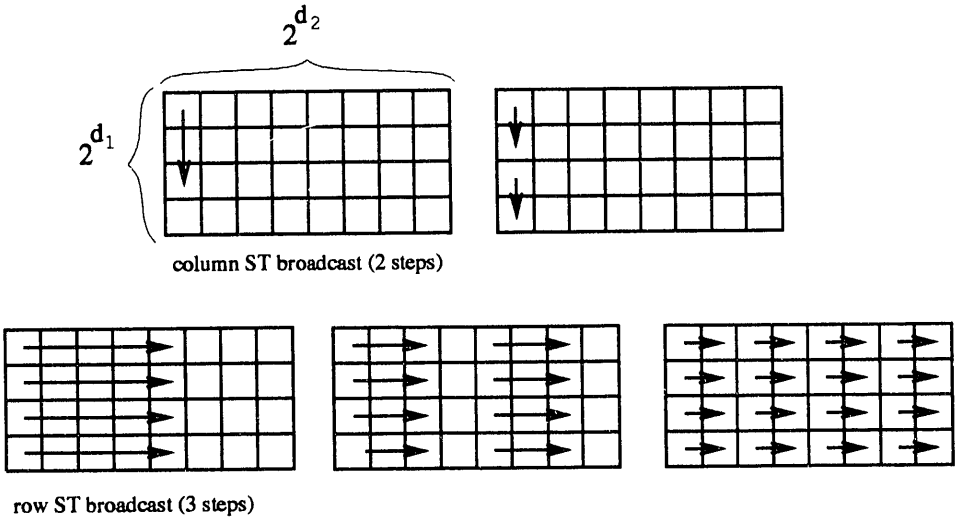


Figure 4: A simple spanning tree broadcast on a mesh.

paragraph to concurrently broadcast the message in its submesh. Steps 3 through 6 of Figure 5 show these broadcasts. All link and port contention can be avoided by alternating the orientation of the spanning trees of the four submeshes. If alternate mesh nodes are colored red and black as in a checkerboard, then the red cells broadcast the message first in their column and then in their row, and the black cells do the opposite. If $d_1 \neq d_2$, the broadcasts along the shorter axis must be delayed for $|d_1 - d_2|$ steps after the two broadcasts along the longer axis begin. The cost of this phase is the cost of the broadcasts along the longer axis,

$$2(\max(d_1, d_2) - 1)(\frac{ma}{4} + b).$$

At the end of the ST broadcasts the packet distribution pattern of the first two steps is now replicated in each contiguous 2×2 block of nodes. Two exchanges between neighboring pairs of nodes complete the algorithm. The cost of these two exchanges is

$$\frac{3}{4}ma + 2b.$$

For a $2^{d_1} \times 2^{d_2}$ mesh the total cost of the algorithm is

$$(\frac{\max(d_1, d_2)}{2} + 1)ma + (2\max(d_1, d_2) + 2)b.$$

For sufficiently long messages, this is a significant improvement over the simple algorithm given at the beginning of this section.

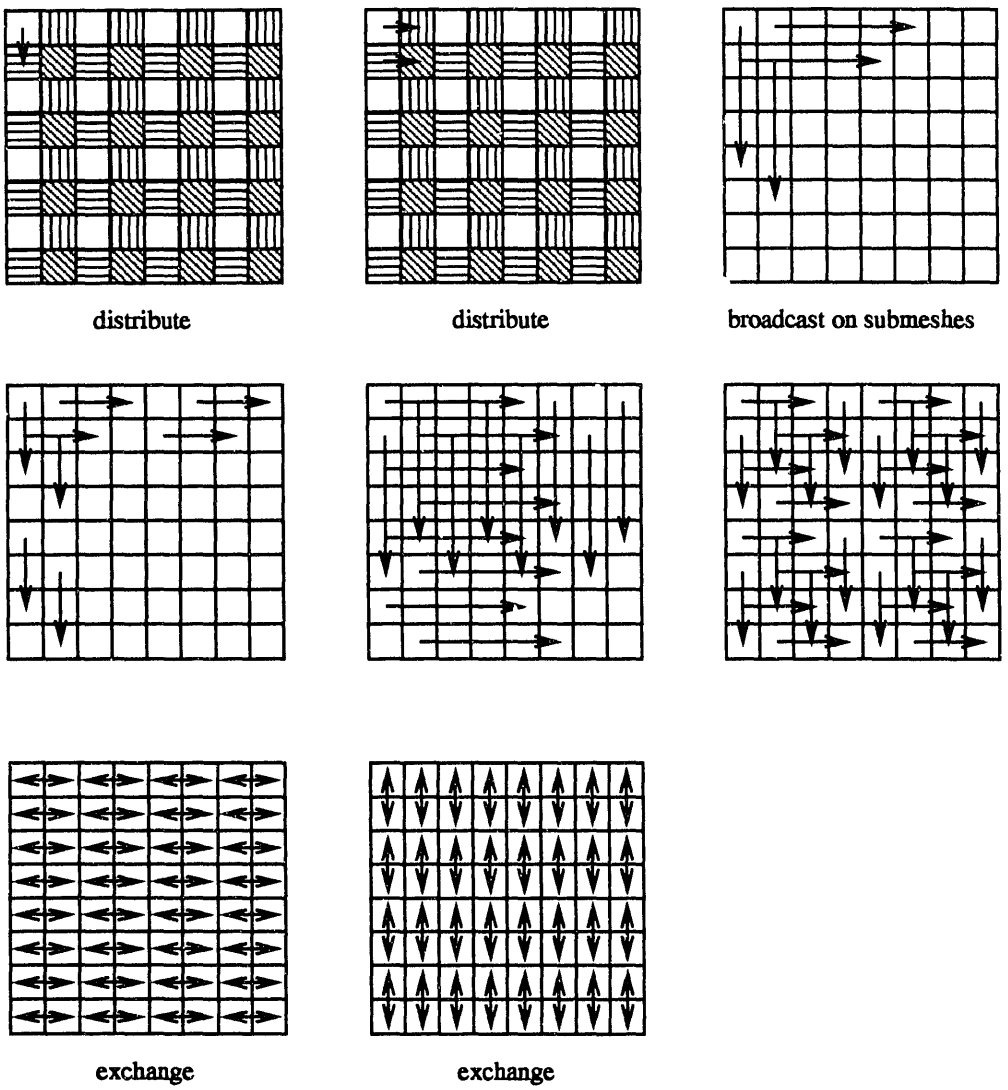


Figure 5: A better ST broadcast on a mesh.

3.2.2. ST broadcast on a mesh, $\nu > 0$

For $\nu > 0$ the ST broadcast algorithm can be generalized from a linear array to a mesh just as in the case of $\nu = 0$, that is, first broadcast along the leftmost column and then concurrently broadcast along each row. It follows from Equation 2 that this broadcast algorithm has cost

$$(4 + \frac{d_1 + d_2 - 2\nu - 4}{2^\nu})ma + (d_1 + d_2 + 2\nu)b. \tag{3}$$

However, some savings can be achieved by delaying the collection step at the end of the column broadcast until the row broadcasts are complete. This way, the packets broadcast along the rows are somewhat shorter. In a mesh of $2^{d_1} \times 2^{d_2}$ nodes, this algorithm proceeds in five phases (see Figure 6):

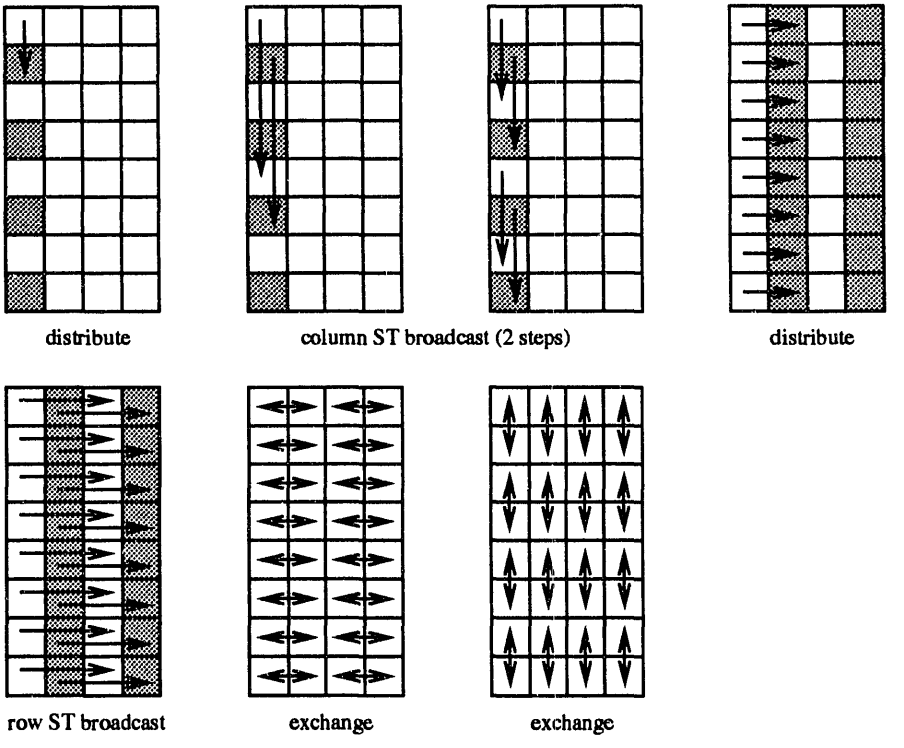


Figure 6: ST broadcast on a mesh, $\nu = 1$.

1. Distribute the message to the first 2^ν nodes in the leftmost column. Cost: $(1-\frac{1}{2^\nu})ma+\nu b$.
2. The first 2^ν nodes in the leftmost column each do a ST broadcast of a message of length $ma/2^\nu$ in a linear subarray (column) of $2^{d_1-\nu}$ nodes. Cost: $(d_1 - \nu)(\frac{ma}{2^\nu} + b)$.
3. Each node in the leftmost column distributes the message it received in phase 2 to the first 2^ν nodes in its row. Cost: $(1 - \frac{1}{2^\nu})\frac{ma}{2^\nu} + \nu b$.

4. The first 2^ν nodes in each row do a ST broadcast of a message of length $ma/2^{2\nu}$ in a linear subarray (row) of $2^{d_2-\nu}$ nodes. Cost: $(d_2-\nu)(\frac{ma}{2^{2\nu}}+b)$.

5. Each contiguous $2^\nu \times 2^\nu$ block of nodes exchanges packets to reconstruct the original message in each node. Cost: $(1-\frac{1}{2^{2\nu}})ma+2\nu b$.

The distribute operations of steps 1 and 5, and the dimensional exchange of step 5, cause no link contention because each of those operations is limited to continguous subarrays of 2^ν nodes or to continguous submeshes of $2^\nu \times 2^\nu$ nodes. Also, it is easy to see that the 2^ν ST broadcasts overlapped in each of steps 2 and 4 do not exceed link capacity. The total cost of this algorithm is thus

$$(3+\frac{d_1-\nu-2}{2^\nu}+\frac{d_2-\nu-1}{2^{2\nu}})ma+(d_1+d_2+2\nu)b. \tag{4}$$

(In view of this cost, when $d_1 > d_2$ it is advantageous to first broadcast along the top row and then down the columns.) This cost is always lower than that of Equation 3. Also note that when $\nu = 0$, this cost becomes $(d_1+d_2)(ma+b)$, as expected.

One additional algorithm is given that trades transmission costs for latency costs. It combines the "better" mesh ST broadcast algorithm given in Section 3.2.1 with the advantage that $\nu > 0$. Again, the mesh is viewed as $2^{2\nu}$ interleaved meshes each of size $2^{d_1-\nu} \times 2^{d_2-\nu}$. The message is first distributed to the block of $2^\nu \times 2^\nu$ nodes in the upper left corner of the mesh at a cost of

$$(1-\frac{1}{2^{2\nu}})ma+2\nu b.$$

Each of the nodes that received a packet then acts as the source of a ST broadcast in its submesh, using the "better" ST broadcast algorithm of Section 3.2.1. Since each packet has length $m/2^{2\nu}$, the cost of this broadcast is

$$(\frac{\max(d_1,d_2)-\nu}{2}+1)\frac{ma}{2^{2\nu}}+(2(\max(d_1,d_2)-\nu)+2)b. \tag{5}$$

At the end of this broadcast each block of $2^\nu \times 2^\nu$ nodes uses a dimensional exchange to collect the entire message into each node. The cost of this collection step is

$$(1-\frac{1}{2^{2\nu}})ma+2\nu b,$$

and the cost of the entire algorithm is

$$(2+\frac{\max(d_1,d_2)-\nu-2}{2^{2\nu+1}})ma+(2\max(d_1,d_2)+2\nu+2)b. \tag{6}$$

Comparing Equations 4 and 6 and assuming that $d_1 \leq d_2$, we see that this algorithm has d_2-d_1+2 more steps, so its latency cost is higher while its transmission cost has been reduced.

This algorithm will be the one chosen for comparison with others in Section 3.7.

3.3. Bidirectional spanning tree broadcast on a linear array

In all of the ST broadcast algorithms for linear arrays presented in Section 3.1, messages flow strictly from left to right when node 0 is the source of the broadcast message. Similarly, in the mesh algorithms of Section 3.2 messages flow only from top to bottom and from left to right. Broadcast algorithms that have bidirectional message flow are now described. These algorithms exploit the network property that messages moving in opposite directions do not contend with each other for communication links. Each of the algorithms presented in Section 3.1 has an analogous bidirectional version, presented in this section. The bidirectional analogs of the ST broadcast algorithms for meshes are given in Section 3.4.

3.3.1. BST broadcast on a linear array, $\nu = 0$

In a linear array of $n = 2^d$ nodes, the *bidirectional spanning tree (BST) broadcast* algorithm broadcasts the message over two spanning trees, one rooted at node 0 and the other rooted at node $2^d - 1$. Node 0 first sends half of the message to the root of the other spanning tree. Both root nodes then do a ST broadcast of their halves of the message over their respective spanning trees. On the last step neighboring pairs of nodes exchange their halves of the message, completing the broadcast. Figure 7 shows the message routing determined by these two trees. There is no link contention because all messages from node 0 are transmitted from left to right

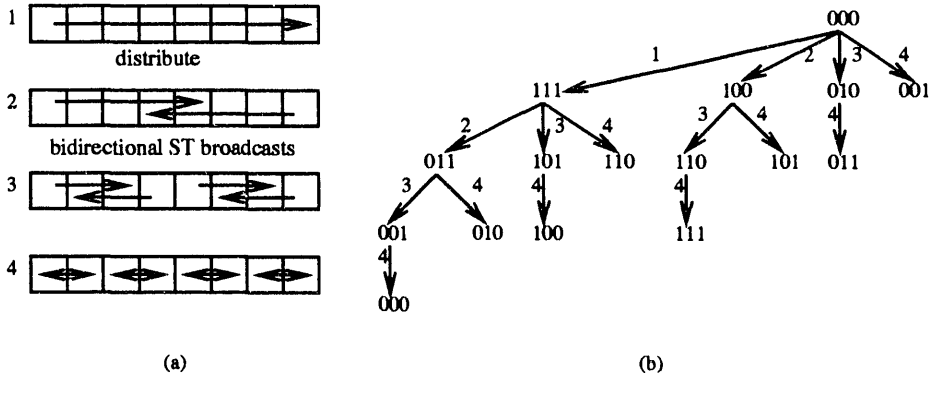


Figure 7: Bidirectional spanning tree broadcast on a linear array.

while all the messages from node $2^d - 1$ are transmitted from right to left. There is no port contention because, until the last step, only even-numbered nodes send and receive messages in the spanning tree rooted at node 0 and only odd-numbered nodes send and receive messages

in the spanning tree rooted at node $2^d - 1$. This algorithm has cost

$$(d+1)(\frac{ma}{2} + b) \quad \text{for a linear array of } 2^d \text{ nodes.} \tag{7}$$

Comparing this cost to the cost of the analogous ST broadcast algorithm from Section 3.1.1 (Equation 1), we see that for $d > 1$, transmission time has been reduced at the cost of one additional startup.

3.3.2. BST broadcast on a linear array, $\nu > 0$

If $\nu > 0$, the linear array can be viewed as 2^ν interleaved subarrays each with $2^{d-\nu}$ nodes, just as in Section 3.1.2. (See Figure 8 and compare with Figure 3.)

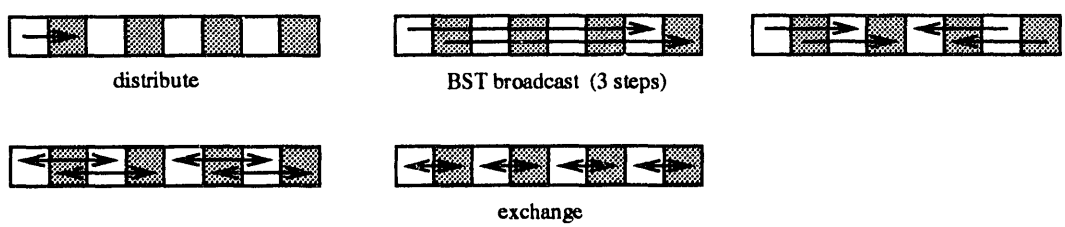


Figure 8: BST broadcast on a linear array, $\nu = 1$.

The message is first distributed among nodes 0, 1,..., $2^\nu - 1$. The cost of this distribution is

$$(1 - \frac{1}{2^\nu})ma + \nu b.$$

Each of the nodes 0, ..., $2^\nu - 1$ then acts as the source node of a BST broadcast of a message of length $m/2^\nu$ over a subarray of $2^{d-\nu}$ nodes, at a cost of

$$(d - \nu + 1)(\frac{ma}{2^{\nu+1}} + b).$$

Since 2^ν BST broadcasts are performed concurrently, the maximum number of messages that contend for a link is 2^ν, so there is no link contention during this phase. There is no port contention because none of the spanning trees have any nodes in common. It is easy to verify that the BST broadcasts leave the packets ordered in each contiguous subarray of 2^ν nodes just as they were in the original partitioning. The final phase of the algorithm consists of combining the packets to reconstruct the original message. This is accomplished by a dimensional exchange within each contiguous subarray of 2^ν nodes. Since these exchanges are localized among subarrays of 2^ν nodes, there is no link contention during this phase. Also, recall that the dimensional exchanges can be done in an ordering that allows the original message to be reconstructed without the need for permuting the packets within each node. Thus, this phase

has cost

$$(1 - \frac{1}{2^\nu})ma + \nu b.$$

The total cost is

$$(2 + \frac{d - \nu - 3}{2^{\nu+1}})ma + (d + \nu + 1)b. \tag{8}$$

When $\nu = 0$ this cost reduces to that of Equation 7, as it should.

3.4. Bidirectional spanning tree broadcast on a mesh

3.4.1. BST broadcast on a mesh, $\nu = 0$

Applying the BST broadcast to a mesh, node 0 first broadcasts the message to the leftmost column of the mesh and then each node in the leftmost column broadcasts to the nodes in its row. If these row and column broadcasts each use the BST broadcast algorithm for a linear array, it follows from Equation 7 that the cost is

$$(d_1 + d_2 + 2)(\frac{ma}{2} + b) \quad \text{for a mesh of } 2^{d_1} \times 2^{d_2} \text{ nodes.}$$

A slight improvement can be obtained by treating the mesh as a linear array of $2^{d_1+d_2}$ nodes and applying the BST algorithm just once to broadcast the message. It is easy to verify that there is no link or port contention in this version of the broadcast. The cost of this approach is

$$(d_1 + d_2 + 1)(\frac{ma}{2} + b). \tag{9}$$

However, neither of these broadcast algorithms take very good advantage of the connectivity of the mesh. A lower cost algorithm can be obtained by an approach analogous to that used in Section 3.2.1 for the "better" ST broadcast on a mesh. The message is first distributed among two square blocks of 4 nodes each at opposite corners of the mesh. The details of this distribution phase are shown in the first three steps of Figure 9. The message is viewed as eight packets numbered $0, 1, ..., 7$. Initially, node $(0, 0)$ contains all eight packets. Distributing the packets as shown in the figure avoids permutation costs later in the algorithm. The cost of this distribution phase is fixed at

$$\frac{7}{8}ma + 3b.$$

Each of the eight nodes that has a packet now uses a spanning tree to concurrently broadcast its packet of length $m/8$ among eight interleaved meshes each of size $2^{d_1-1} \times 2^{d_2-1}$. Steps 4 through 7 of Figure 9 show these broadcasts. All link and port contention is avoided by alternating the orientation of those spanning trees as described in Section 3.2.1. (Also, compare with Figure 5.) If $d_1 \neq d_2$, the broadcasts along the shorter axis must be delayed for $|d_1 - d_2|$ steps after each

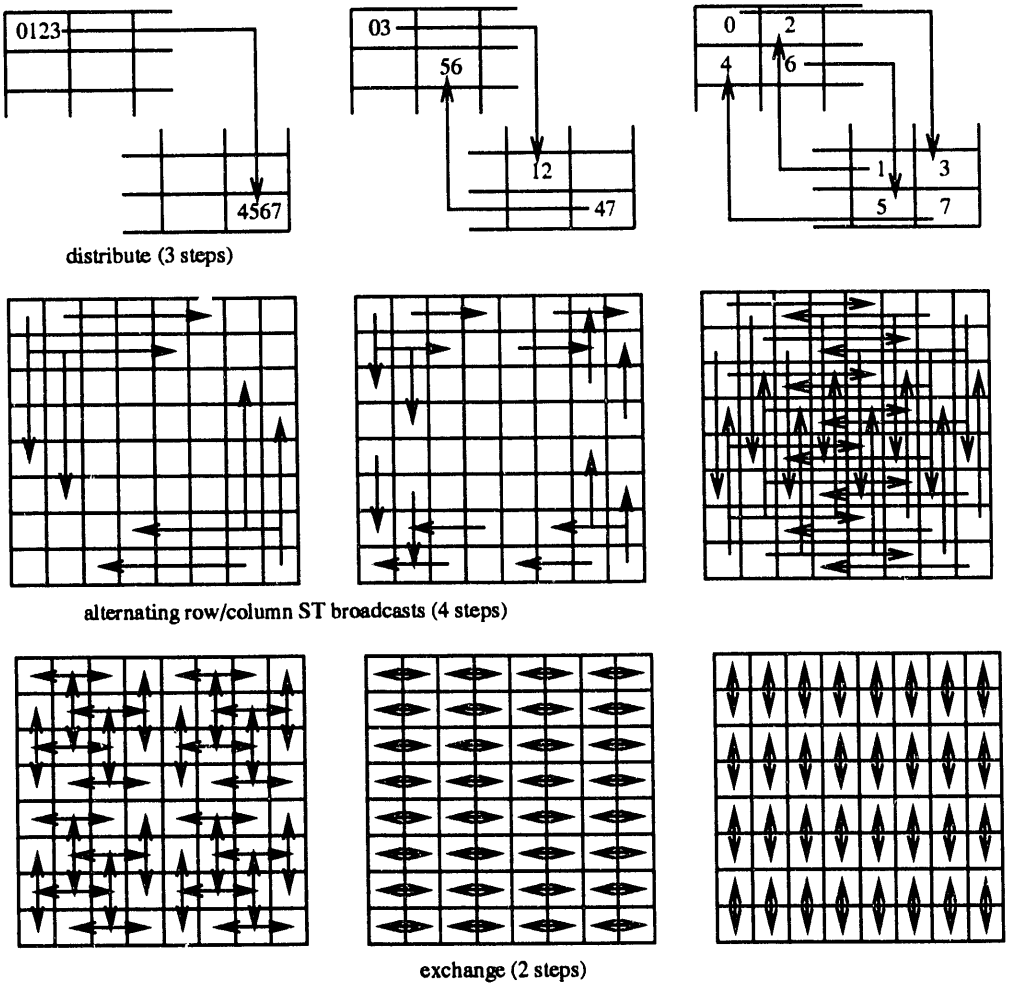


Figure 9: A better BST broadcast on a mesh.

of the two broadcasts along the longer axis begin. The cost of this phase is the cost of the broadcasts along the longer axis,

$$2(\max(d_1, d_2) - 1)(\frac{ma}{8} + b).$$

At the end of the ST broadcasts, each continguous square block of four nodes contains the entire message, distributed among its members. Figure 10 shows how the original eight packets are distributed among the members of each such block. This particular packet distribution pattern

	$2j$	$2j+1$
$2i$	01	23
$2i+1$	45	67

Figure 10: Packet distribution after broadcast phase. Row and column indices indicate block orientation. ($0 \leq i < 2^{d_1-1}$ and $0 \leq j < 2^{d_2-1}$.)

allows the original message to be reconstructed in each node, without permutation costs, by the pair of exchanges shown in the last two steps of Figure 9 and with a cost of

$$\frac{3}{4}ma + 2b.$$

The total cost of the algorithm is thus

$$\frac{2\max(d_1, d_2) + 11}{8}ma + (2\max(d_1, d_2) + 3)b. \tag{10}$$

3.4.2. BST broadcast on a mesh, $\nu > 0$

As in Section 3.2.2, consider the $2^{d_1} \times 2^{d_2}$ mesh as made up of $2^{2\nu}$ interleaved submeshes, each of size $2^{d_1-\nu} \times 2^{d_2-\nu}$, so that each $2^\nu \times 2^\nu$ contiguous block of nodes has exactly one node from each submesh. To broadcast a message, node 0 first distributes the message as $2^{2\nu}$ packets among the nodes in the $2^\nu \times 2^\nu$ block in the upper left corner of the mesh. The cost of this step is

$$(1 - \frac{1}{2^{2\nu}})ma + 2\nu b.$$

Each of the nodes in that block then acts as the root of a BST broadcast of a message of length $m/2^{2\nu}$ in a submesh of size $2^{d_1-\nu} \times 2^{d_2-\nu}$ with $\nu = 0$. From Equation 10, the cost of those broadcasts is

$$\frac{2(\max(d_1, d_2) - \nu) + 11}{2^{2\nu+3}}ma + (2(\max(d_1, d_2) - \nu) + 3)b.$$

After those broadcasts are completed, each node in each contiguous $2^\nu \times 2^\nu$ block contains one of the $2^{2\nu}$ packets. These packets are then recombined using a dimensional exchange. The ordering of these packets within each block is the same as the ordering of the packets after the distribution phase so there are no permutation costs during the exchange. Also, the level of link contention during the exchange is never greater than the network's capacity to handle it because communication is localized within blocks of $2^\nu \times 2^\nu$ nodes. This dimensional exchange thus has cost

$$(1 - \frac{1}{2^{2\nu}})ma + 2\nu b$$

and so the total cost of this algorithm is

$$(2 + \frac{2\max(d_1, d_2) - 2\nu - 5}{2^{2\nu+3}})ma + (2\max(d_1, d_2) + 2\nu + 3)b.$$

3.5. Recursive halving broadcast on a linear array

The *recursive halving (RH) broadcast* is similar to the recursive halving broadcast algorithm for hypercubes given by van de Geijn and it differs only slightly from the global combine algorithm for linear arrays and meshes given in [1,16].

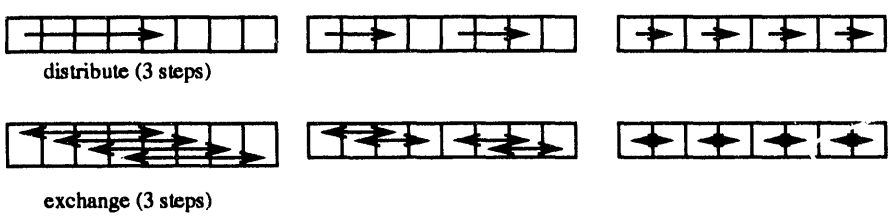


Figure 11: Recursive halving broadcast on a linear array.

The first phase of the RH broadcast uses a spanning tree to distribute the message among all processors. This is shown in the first three steps of Figure 11. The cost of the first phase is

$$(1 - \frac{1}{2^d})ma + db.$$

The second phase recombines the message packets using a sequence of pairwise exchanges analogous to a dimensional exchange in a hypercube. On step i ($1 \leq i \leq d$), all pairs of nodes whose $(d-i)^{th}$ bits differ exchange messages. With each exchange the lengths of the message packets double until each node contains the entire message after d steps. There is link contention at each step of this phase but the last. The amount of contention decreases with each step. On the first step 2^{d-1} messages contend for the link from node $2^{d-1}-1$ to node 2^{d-1}. The same number of messages contend for the link going in the other direction. If $\nu > 0$, the link contention that occurs during the second phase is mitigated somewhat by the facts that

the shortest messages are sent during steps having the greatest link contention, and, if $\nu > 0$ the bandwidth of the network links is a factor of 2^ν higher than that of the connection from the node to the network (since $a = 2^\nu \overline{a}$), so 2^ν pairs of exchanges can proceed concurrently. Under these considerations the cost of the second phase of the RH broadcast is

$$\sum_{i=0}^{d-1} [\frac{m}{2^{i+1}} \max(a, 2^i \overline{a}) + b] \quad = \quad (1 + \frac{d-\nu-2}{2^{\nu+1}})ma + db.$$

This algorithm partitions the original message into 2^d packets. An examination of the routes followed by those packets shows that in order for the broadcast algorithm to preserve their original ordering they must either be permuted in node 0 before the first phase or permuted in each node following the second phase. Either choice has cost $m\rho$. The total cost of the RH broadcast is thus

$$(2 + \frac{d-\nu-2}{2^{\nu+1}} - \frac{1}{2^d})ma + 2db + m\rho \quad \text{for a linear array of } 2^d \text{ nodes.}$$

3.6. Recursive halving broadcast on a mesh

The RH broadcast algorithm for a mesh is similar to the RH broadcast algorithm for a linear array. The cost of distributing the message in the first phase (shown in the first six frames of Figure 12) is

$$(1 - \frac{1}{2^{d_1+d_2}})ma + (d_1 + d_2)b.$$

The dimensional exchange of the second phase can take advantage of both dimensions of the mesh to reduce contention. By interleaving horizontal and vertical exchanges in alternate nodes (shown in the remaining six frames of Figure 12) the amount of contention that occurs in a square mesh ($d_1 = d_2$) is half that which would occur if all horizontal exchanges were done before any vertical exchanges. For non-square meshes the reduction is not as great. (See [1] for further details.) For $\nu < d_1 \leq d_2$, the cost of the second phase is

$$\sum_{i=0}^{2d_1-1} [\frac{m}{2^{i+1}} \max(a, 2^{\lfloor i/2 \rfloor - 1} \overline{a}) + b] + \sum_{i=2d_1}^{d_1+d_2-1} [\frac{m}{2^{i+1}} \max(a, 2^{i-d_1} \overline{a}) + b]$$

$$= (1 + \frac{1}{2^{2\nu+3}} + \frac{2(d_2-d_1)-3}{2^{d_1+\nu+2}})ma + (d_1 + d_2)b.$$

As in a RH broadcast on a linear array, permutation costs of $m\rho$ are incurred at the beginning or at the end of the algorithm, so the total cost of the mesh RH broadcast for $2^{d_1} \times 2^{d_2}$ nodes is

$$(2 + \frac{2(d_2-d_1)-3}{2^{d_1+\nu+2}} + \frac{1}{2^{2\nu+3}} - \frac{1}{2^{d_1+d_2}})ma + 2(d_1 + d_2)b + m\rho.$$

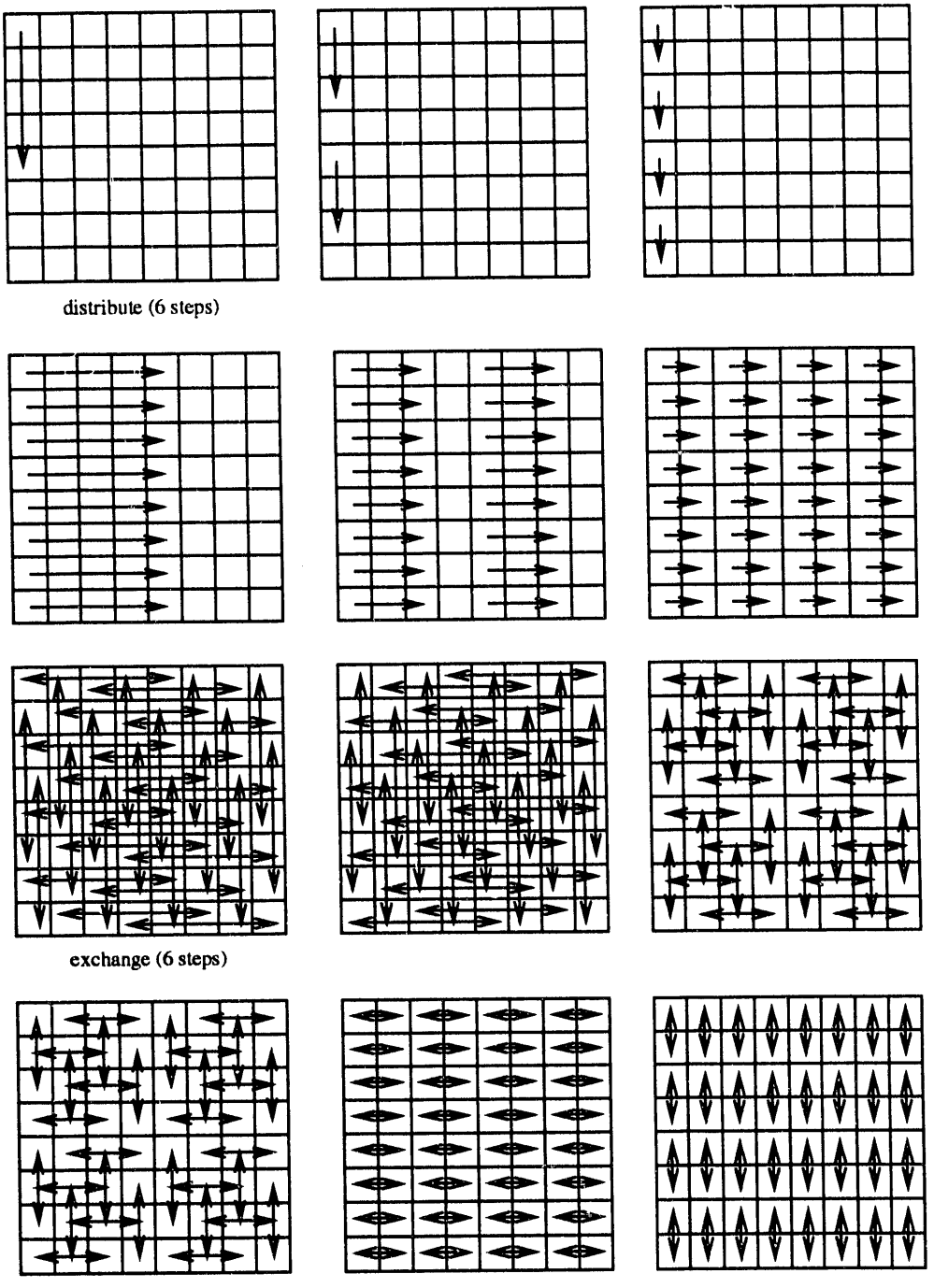


Figure 12: RH broadcast on a mesh.

3.7. Comparisons

Table I summarizes the costs of the broadcast algorithms described in this section. The costs of the algorithms designed for linear arrays are given for $n = 2^d$ nodes while the costs of the mesh algorithms are given for $n = 2^{d_1} \times 2^{d_2}$ nodes. It is assumed that $\nu < d$ and $\nu < d_1 \leq d_2$. All of these algorithms have $O(\lg n)$ coefficients of transmission cost and latency. The RH broadcast has an $O(1)$ coefficient of transmission cost in the special case of square meshes ($d_1 = d_2$).

linear ST	$(2 + \frac{d-\nu-2}{2^\nu})ma + (d+\nu)b$
linear BST	$(2 + \frac{d-\nu-3}{2^{\nu+1}})ma + (d+\nu+1)b$
linear RH	$(2 + \frac{d-\nu-2}{2^{\nu+1}} - \frac{1}{2^d})ma + 2db + m\rho$
mesh ST	$(2 + \frac{\max(d_1,d_2)-\nu-2}{2^{2\nu+1}})ma + (2\max(d_1,d_2) + 2\nu + 2)b$
mesh BST	$(2 + \frac{2\max(d_1,d_2)-2\nu-5}{2^{2\nu+3}})ma + (2\max(d_1,d_2) + 2\nu + 3)b$
mesh RH	$(2 + \frac{2(d_2-d_1)-3}{2^{d_1+\nu+2}} + \frac{1}{2^{2\nu+3}} - \frac{1}{2^{d_1+d_2}})ma + 2(d_1+d_2)b + m\rho$

Table I: Broadcast algorithm costs.

For the shortest messages the ST broadcast algorithms have the lowest cost on both linear arrays and meshes because of their low latency. The BST broadcast always has lower cost than the RH broadcast on linear arrays, regardless of message length. The BST broadcast also has lower cost than the ST broadcast whenever

$$m > \frac{2^{\nu+1}b}{a(d-\nu-1)}.$$

It can be verified from the data given in Section 6 that this crossover point is reached on the Intel Delta mesh before $m >$ 1K bytes.

The costs of the ST and BST broadcast algorithms for meshes in Table I are those of the "better" versions of those algorithms given at the ends of Section 3.2.2 and 3.4.2, respectively. The mesh BST broadcast algorithm has one more step than the mesh ST broadcast algorithm and so its latency cost is higher by b. The difference between the coefficients of ma in the cost expressions for those two algorithms is not as great as in the case of the algorithms for linear arrays and so the additional latency of the BST broadcast is not as quickly amortized by longer message lengths. On the Delta the predicted performance of the BST broadcast is better than that of the ST broadcast only for messages of many tens of kilobytes. The value of the crossover point varies inversely with the size of the mesh. The RH broadcast algorithm for meshes sometimes has lower transmission cost than the ST and BST broadcast algorithms,

but the latter always have lower latency costs and they have no permutation costs.

Finally, note that the ST and BST broadcast algorithms require no additional temporary storage of message packets because they do not need to permute message packets. The RH broadcast algorithms require temporary storage proportional to message length.

4. Broadcasting from an Arbitrary Node

In all of the broadcast algorithms given so far, the node that contains the message to be broadcast has always been node 0 in the case of linear arrays, and node (0,0) in the case of meshes. This section considers the problem of generalizing the algorithms of Section 3 so that any node can serve as the source of the broadcast message. This is done for the ST and BST broadcasts on linear arrays and for the RH broadcast on linear arrays and meshes. None of these generalizations adds any cost to the original algorithms. It is conjectured that a similar approach can be used to generalize the ST and BST broadcast algorithms for meshes.

4.1. ST broadcast on a linear array, $\nu = 0$

It was mentioned in Section 3.1.1 that there is no link contention during a ST broadcast on a linear array. This is clear from Figure 1(a) when node 0 is the root of the spanning tree. In fact, there is no link contention even when some other node is chosen as the root of the ST broadcast. It is now shown that for a ST broadcast from an arbitrary node of a linear array of $n = 2^d$ nodes, at each step each message travels the same distance and in the same direction, and all nodes that send messages are separated by a distance that is greater than the distance their messages travel. This will allow us to conclude that there is no link contention during such a broadcast.

Consider a linear array with $n = 2^d$ nodes and suppose node k is the source of the broadcast message. Construct the spanning tree for the broadcast by exclusive OR-ing k with the node numbers of the spanning tree with root 0. (See Figure 13(b).) Now assume that at the beginning of the i^{th} step of the broadcast, any two nodes that have a copy of the message are separated by a distance of at least 2^{d-i+1} in the linear array. (This is vacuously true at the beginning of step 1.) Each message sent during the i^{th} step travels a distance of 2^{d-i} hops because it is the $(d-i)^{\text{th}}$ bit of the sender's node number that is complemented to determine the destination of the message. In addition, each message sent during the i^{th} step travels in the same direction because the i^{th} bit of each sender is the same. This is clearly true in the spanning tree with root 0; all such bits are 0 and all messages travel to the right at the i^{th} step. This is also true in the spanning tree rooted at node k since the $(d-i)^{\text{th}}$ bits of the sending nodes were all obtained by exclusive OR-ing the $(d-i)^{\text{th}}$ bit of k with 0, so at the end of step i, all nodes that have a copy of the message are separated by a distance of at least 2^{d-i}. Thus, at each step all

message follow disjoint paths and so there is no link contention. It follows that there is no link contention during an ST broadcast from an arbitrary node of a linear array of 2^d nodes. (See Figure 13(a).)

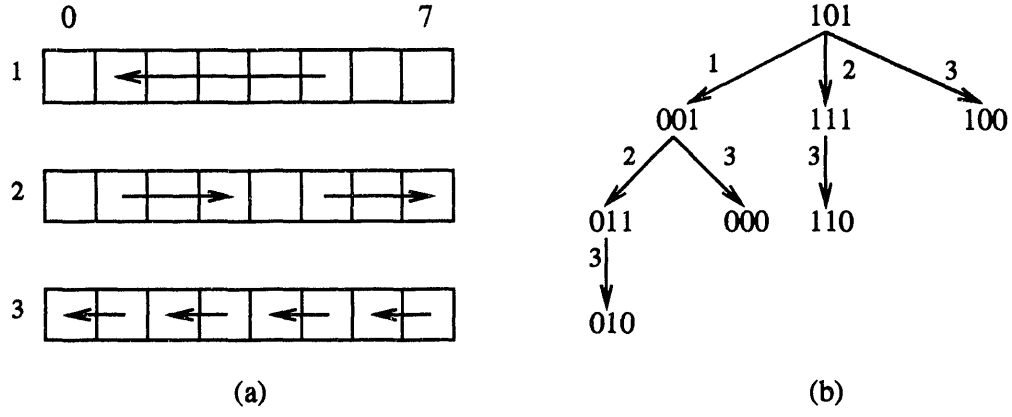


Figure 13: Spanning tree broadcast on a linear array with root node 5.

Based on these observations it is also clear that there is no port contention during a ST broadcast from an arbitrary node because at each step each message has a distinct destination. Also, there is no link or port contention during a distribution operation from an arbitrary node. This follows because the distribution algorithm uses the same spanning tree as the ST broadcast. These observations are now applied to generalize some of the other broadcast algorithms presented in Section 3.

4.2. ST broadcast on a linear array, $\nu > 0$

First, consider the ST broadcast algorithm for linear arrays with $\nu > 0$. (See Section 3.1.2 and Figure 3.) That algorithm has three phases consisting of a distribute operation, concurrent ST broadcasts on subarrays (with $\nu = 0$), and a dimensional exchange. During the first phase the message is distributed to a contiguous block of 2^ν nodes. When node 0 is the source of the broadcast this block consists of nodes $0, 1, ..., 2^\nu - 1$. When some other node k is the source the message should be distributed among the corresponding block of nodes that contains node k, namely, nodes $j2^\nu, j2^\nu + 1, ..., (j+1)2^\nu - 1$, where $0 \leq j < 2^{d-\nu}$ and $j2^\nu \leq k \leq (j+1)2^\nu - 1$. Call this block of nodes *metanode* j. If node k is not the leftmost node in metanode j the distribute algorithm must use a spanning tree rooted at some node other than the leftmost node in the metanode. It was pointed out above that there is no link or port contention in such a distribute operation. Also note that it is a simple matter to consecutively distribute the 2^ν packets among the nodes of the metanode, so no permutation costs are introduced.

During the next phase there are 2^ν concurrent ST broadcasts performed on 2^ν interleaved

subarrays. Note that at each step of the concurrent ST broadcasts, all messages sent by nodes in metanode j have destinations in the same metanode. Figure 13(a) can thus be viewed as a ST broadcast over a linear array of metanodes so the argument given in the case of $\nu = 0$ applies equally well to the case of $\nu > 0$.

The final phase of the ST broadcast is the dimensional exchange of packets among nodes in the same metanode. At this point the source node of the broadcast is irrelevant since each metanode contains the same information, so the issues of link and port contention do not arise. It follows from these observations that there is no link or port contention during a ST broadcast from an arbitrary node for linear arrays with $\nu > 0$. It is conjectured that similar arguments can be used to extend these observations to the mesh ST broadcast algorithms given in Section 3.2.

4.3. BST broadcast on a linear array

Now consider the BST broadcast on a linear array with $\nu = 0$. (See Section 3.3.1 and Figure 7.) If some node $k \neq 0$ is the source of the broadcast message, then the appropriate broadcast tree is obtained by exclusive OR-ing k with each node of the BST broadcast tree with root 0. (See Figure 14.) On the first step the message is divided into two halves. Node k sends one half

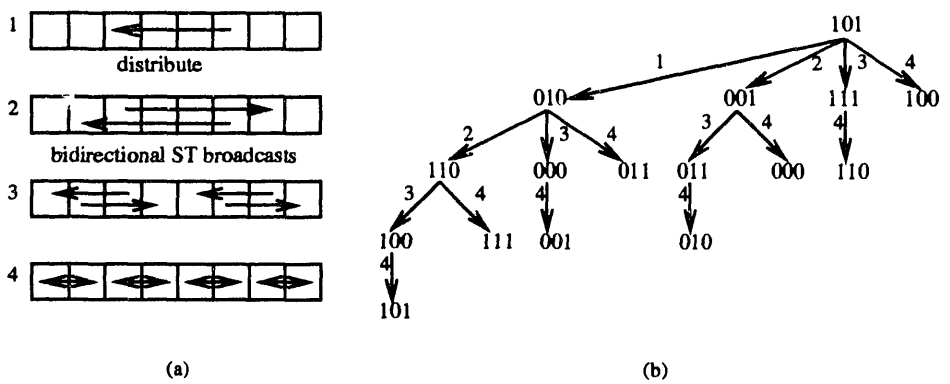


Figure 14: BST broadcast on a linear array with root node 5.

to node $\overline{k}$, that is, to the bit-wise complement of k. Nodes k and $\overline{k}$ then act as source nodes in ST broadcasts on two interleaved arrays each with 2^{d-1} nodes. It was shown in Section 4.1 that there is no link or port contention during either one of those ST broadcasts. There is no port contention between those two concurrent ST broadcasts because their spanning trees are disjoint, and there is no link contention because at each step all messages in one spanning tree travel in the opposite direction from all message in the other spanning tree. It follows that there is no link or port contention during a BST broadcast on a linear array with $\nu = 0$. This observation can be extended to the case of $\nu > 0$ by viewing a BST broadcast on a linear array of 2^d nodes with $\nu > 0$ to be a BST broadcast on a linear array of $2^{d-\nu}$ metanodes with $\nu = 0$.

The reasoning is the same as in the case of a ST broadcast with $\nu > 0$, above. It is conjectured that similar arguments can be used to extend these observations to the mesh BST broadcast algorithms given in Section 3.4.

4.4. RH broadcasts on linear arrays and meshes

Finally, the RH broadcast algorithm for linear arrays can be easily generalized to start from an arbitrary node. The first phase of that algorithm is a distribute operation (across the entire array) based on a spanning tree. (See Figure 11.) It was shown above that such a distribute operation can be generalized to start from any node. After the distribute is complete the location of the source node is no longer relevant and the final dimensional exchange phase of the algorithm proceeds in the same way regardless of which node started the broadcast. Similar observations serve to generalize the RH broadcast on a mesh. The first phase is a column distribute followed by a row distribute. (See Figure 12.) These operations are easily extended to start from any node. The remainder of the algorithm consists of exchange operations, and these are not affected by the location of the source node.

5. Broadcasting to Arbitrary Numbers of Nodes

The problem of broadcasting on linear arrays and meshes of arbitrary size is now considered. Two approaches to this problem are described: virtual nodes and companions. The method of virtual nodes can be applied only to the ST and BST broadcast algorithms. The method of companions can be used to generalize all of the broadcast algorithms given in Section 3. The method of companions is also likely to be useful for extending the applicability of other communication algorithms, such as the complete exchange algorithms of [6] and the global combine algorithms of [1], but such considerations are beyond the scope of this work.

5.1. Virtual nodes

Given a linear array of n nodes, append $2^{\lceil \lg n \rceil} - n$ *virtual nodes* to the right end of the array to bring the number of nodes up to the next closest power of two. Figure 15(a) shows an 11-node linear array with five virtual nodes appended to it. The idea is to execute an algorithm designed for $2^{\lceil \lg n \rceil}$ nodes on n nodes, with node $n-1$ simulating all of the virtual nodes to its right. This can only be done if there is not too much message traffic passing through node $n-1$.

For example, suppose $n = 5$ and suppose the rightmost three nodes are virtual nodes in the RH broadcast shown in Figure 11. Then at the fourth step, node 4 would have to send and receive messages from nodes 0, 1, 2, and 3. Since this would increase the cost of the broadcast over the usual cost of a RH broadcast on $2^{\lceil \lg n \rceil}$ nodes, generalizing such algorithms using virtual nodes is not considered worthwhile.

On the other hand, the ST and BST broadcast algorithms for linear arrays with $\nu = 0$ do not require node $n-1$ to send and receive more than one message at a time. These two algorithms can be generalized using virtual nodes. The cost of broadcasting in a row of n nodes is then just the cost of broadcasting in a row of $2^{\lceil \lg n \rceil}$ nodes.

In the case of an $n_1 \times n_2$ mesh, shown in Figure 15(b), each cell in the rightmost column of physical nodes simulates the virtual nodes to its right. Similarly, the bottom row of physical nodes simulates the virtual nodes below. This leaves node (n_1-1, n_2-2), the bottom rightmost physical node, to simulate a submesh of $(2^{\lceil \lg n_1 \rceil} - n_1 + 1)(2^{\lceil \lg n_2 \rceil} - n_2 + 1) - 1$ nodes, shown with double hatching in Figure 15(b). This simulation is feasible for the "simple" ST and the BST broadcast algorithms for meshes with $\nu = 0$ given in Sections 3.2.1 and 3.4.1 because at each step of those algorithms all messages travel only horizontally or only vertically, so node $(n_1 - 1, n_2 - 2)$ has no greater responsibility for the simulation than any other nodes in its row and column. In these two cases the cost of using virtual nodes is the same as the cost of a broadcast on a mesh of $2^{\lceil \lg n_1 \rceil} \times 2^{\lceil \lg n_2 \rceil}$ nodes.

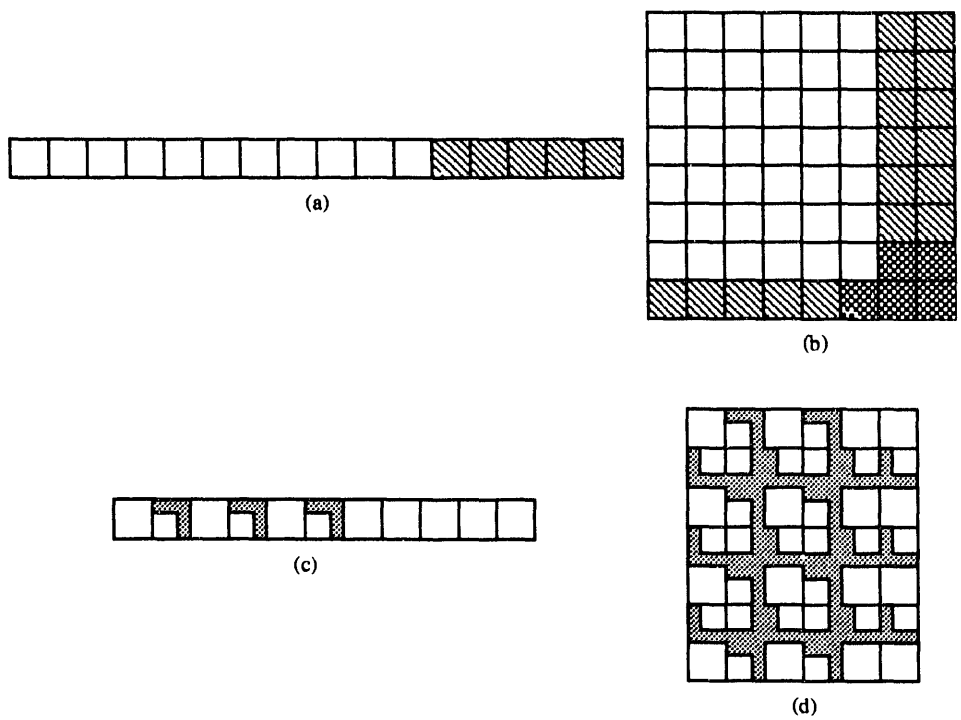


Figure 15: Virtual nodes (a and b), and companions (c and d).

5.2. Companions

The method of companions can be applied uniformly to generalize all of the algorithms of Section 3. Consider the 11-node linear array shown in Figure 15(c). Use any broadcast algorithm

for linear arrays to broadcast the message to the 8 nodes shown full-sized in that figure. Each full-sized node then sends the message to its smaller neighbor, its *companion* (if any), to its immediate right. In a linear array of n nodes, $c = n - 2^{\lfloor \lg n \rfloor}$ nodes have companions. It is assumed that the companion nodes are the first c odd-numbered nodes in the array, the smaller nodes shown in Figure 15(c). The cost of the broadcast is then the cost of broadcasting the message to $2^{\lfloor \lg n \rfloor}$ nodes plus an additional cost of $ma + b$ to send the message to the companions. In an $n_1 \times n_2$ mesh entire rows and columns may consist only of companions, and some nodes in a mesh may have as many as three companions, as shown in Figure 15(d). In this case the cost of the broadcast is that of a broadcast on a mesh of $2^{\lfloor \lg n_1 \rfloor} \times 2^{\lfloor \lg n_2 \rfloor}$ nodes plus $\min(2(ma + b), 3((ma/2) + b))$. The additional term is the minimum of the costs of ST and BST broadcasts, respectively, on a 2×2 mesh. (The RH broadcast algorithm always performs less well than either of these on a 2×2 mesh.) Note that the transmission cost in the additional term is at least $3ma/2$. Also note that the transmission costs of all of the mesh broadcast algorithms given in Table I are approximately $2ma$. This means that when companions are used to generalize those broadcast algorithms, the transmission cost of broadcasting on a mesh of arbitrary size is almost twice that of broadcasting on a mesh with the next smaller powers of 2 rows and columns.

5.3. Comparisons

Only a few of the broadcast algorithms given in Sections 3 can be generalized using virtual nodes. These are the ST and BST broadcast algorithms for linear arrays with $\nu = 0$ given in Sections 3.1.1 and 3.3.1, and the "simple" ST and BST broadcast algorithms for meshes with $\nu = 0$ given in Sections 3.2.1 and 3.4.1. It is easy to verify that generalizing the "simple" mesh broadcast algorithms using virtual nodes yields the same or lower cost than generalizations based on companions, but in both cases the overall lowest costs are obtained by generalizing the "better" mesh broadcast algorithms using companions, so it is not worthwhile to use virtual nodes to generalize mesh algorithms.

For the ST broadcast algorithm for linear arrays with $\nu = 0$, the cost of the algorithm generalized using virtual nodes is $\lceil \lg n \rceil (ma + b)$ and the cost of generalizing with companions is $(\lfloor \lg n \rfloor + 1)(ma + b)$. These two costs are the same when n is not a power of 2. On the other hand, for the BST broadcast algorithm for linear arrays with $\nu = 0$, the cost of the algorithm generalized using virtual nodes is $(\lceil \lg n \rceil + 1)(ma/2 + b)$ and the cost of generalizing with companions is $(\lfloor \lg n \rfloor + 1)(ma/2 + b) + ma + b$. It is easy to check that the former cost is lower. This is the only known case where generalizing a broadcast algorithm using virtual nodes has lower cost than generalizing that algorithm using companions.

6. Performance on the Intel Delta Mesh

The predicted and actual performance of the broadcast algorithms of Section 3 are now considered. Predicted costs are based on the communication model of the Intel Delta mesh presented in Section 2 and on measurements of the constants for that model. The actual costs of the ST, BST and RH algorithms measured on a linear array of 16 nodes are also presented.

It is claimed by Barnett et al. in [1] that $\nu \approx 1$ for the Delta. More recent results given in [13] reveal that this is an accurate estimate of ν only when certain programming techniques are used. Under ordinary circumstances it is more often the case that $\nu \approx 0$ and so it is assumed here that $a = \overline{a}$. The values given in Table II for a, $\overline{a}$, and b are taken from [7]. These performance measurements were made using the "robust" kernel. The remaining constant used in the communication model of the Delta is ρ, the cost of data movement within a node. A careful consideration of this cost for the i860 processing nodes used on the Delta is beyond the scope of this work. The performance measurements given in [4] indicate that a data transfer rate of 100MB/sec is easily obtainable in practice, so ρ is chosen to be 0.01 microseconds per byte.

node-to-network trans. cost	a	$0.08\mu sec/byte$
network-to-network trans. cost	$\overline{a}$	$0.08\mu sec/byte$
latency	b	$75\mu sec$
permutation cost	ρ	$0.01\mu sec/byte$

Table II: Communication constants for the Intel Delta mesh.

The predicted costs of broadcast algorithms are shown in Figure 16 for a linear array of 2^4 nodes. That figure illustrates that the BST broadcast is always faster than the RH broadcast and that the BST broadcast is faster than the ST broadcast on all but the shortest messages.

Figure 17 shows the observed costs of implementations of the ST, BST, and RH broadcast algorithms for a linear array of 16 nodes. Those data confirm that the BST broadcast algorithm performs better than the ST broadcast algorithm for all but the shortest messages. The crossover point is between 256 and 512 bytes. The observed cost of the RH broadcast algorithm was much higher than predicted by the model. This is attributed to the facts that the RH broadcast recursively breaks the original message into packets of length m/n and that the cost model of the Delta used here is not accurate for such packet lengths. This observation about the cost model is supported in [10]. Some improvement in the performance of the RH broadcast can probably be obtained by applying programming techniques described in [13].

The predicted costs of the broadcast algorithms for meshes given in Section 3 are shown in Figure 18 for a $2^4 \times 2^5$ mesh with $\nu = 0$. The predicted costs given for the ST and BST

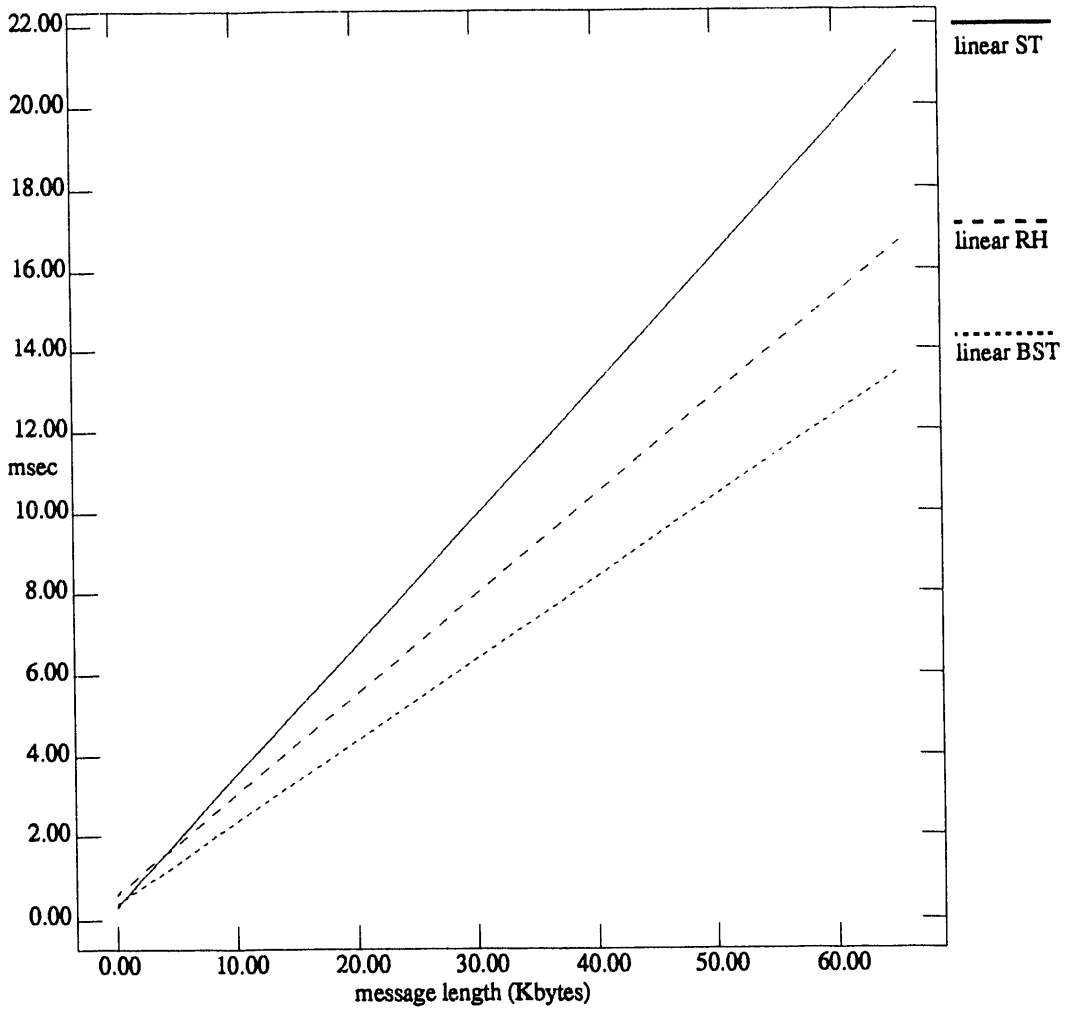


Figure 16: Predicted broadcast costs: $n = 2^4$, $8 \leq m \leq 64\text{K}$, $\nu = 0$.

broadcast algorithms are those of the "better" versions of those algorithms given at the ends of Section 3.2.2 and 3.4.2, respectively. Again, the ST broadcast performs best for short messages and gives way to the BST broadcast for message of between 512 and 1K bytes. The predicted performance of the RH broadcast is better than that of the BST broadcast for messages of more than 12K bytes. Actual performance figures for these algorithms are not available at this time. Work is currently underway to implement these algorithms and measure their costs on the Delta.

The observations made in this section are preliminary. Further work will be conducted to obtain a larger collection of performance results from the Intel Delta and to better determine which features are needed in the communication model to provide accurate predictions of algorithm performance.

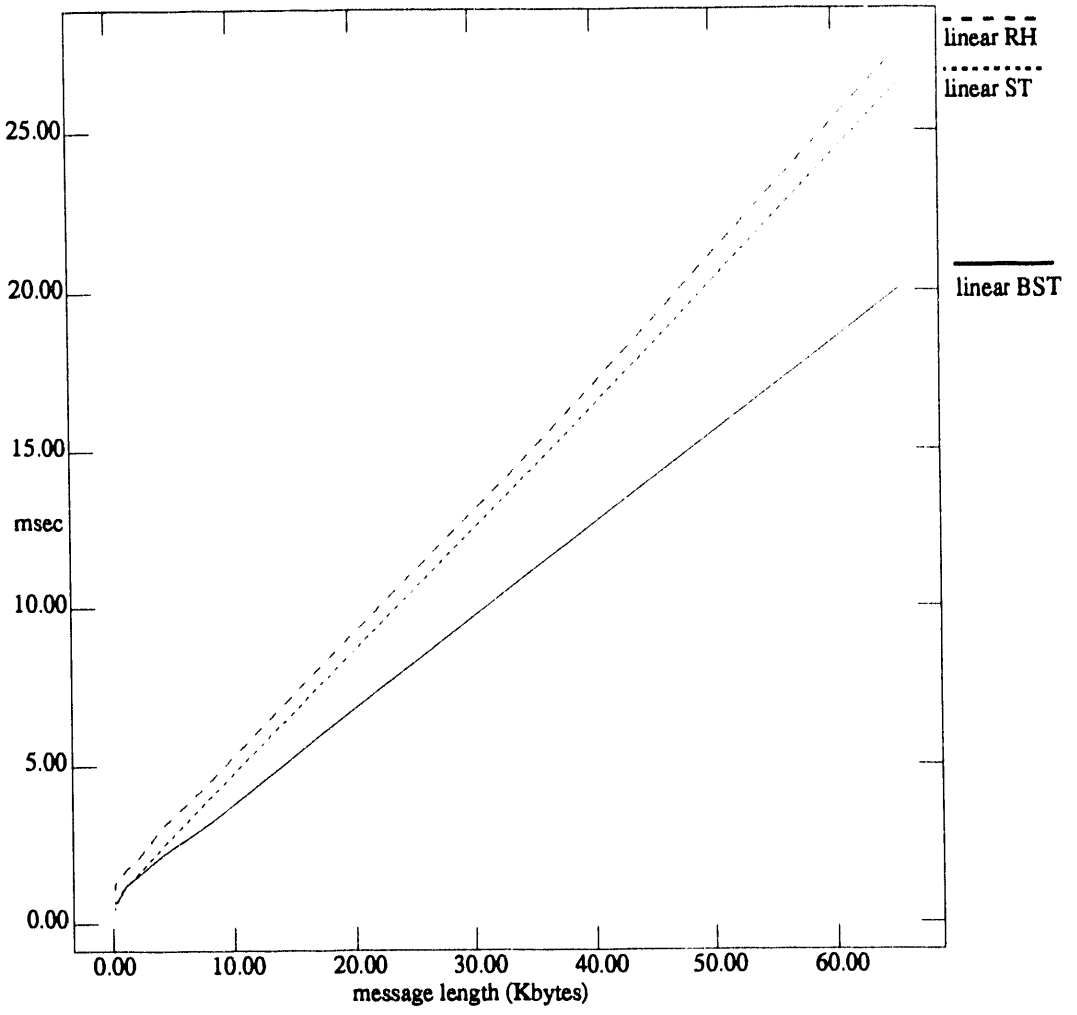


Figure 17: Intel Delta observed broadcast costs: $n = 2^4$, $8 \leq m \leq 64\text{K}$.

7. Summary

This study of the problem of broadcasting on linear arrays and meshes has yielded several improved algorithms. The well known spanning tree (ST) broadcast algorithm for linear arrays was extended in several ways. First, it was modified to take advantage of the difference between node-to-network and network-to-network communication rates. Second, it was modified to take advantage of bidirectional communication. Combining these two extensions of the basic ST broadcast algorithm yielded the bidirectional spanning tree (BST) broadcast algorithm for linear arrays. The BST broadcast algorithm always has lower cost than that of the recursive halving (RH) broadcast algorithm and has lower cost than the ST broadcast algorithm for all but the shortest messages.

Similar consideration was given to the ST broadcast algorithm for meshes. In this case the

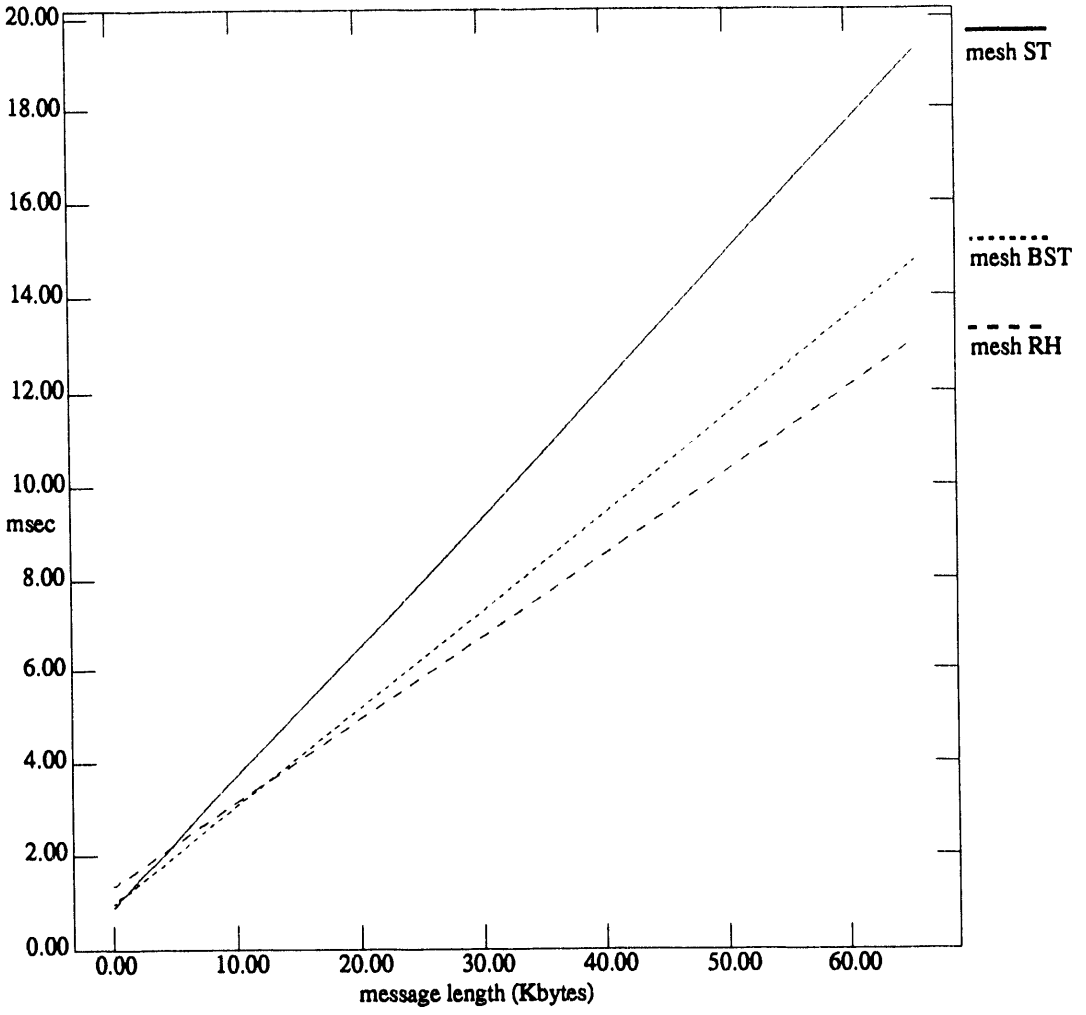


Figure 18: Predicted broadcast costs: $n = 2^4 \times 2^5$, $8 \leq m \leq 64\text{K}$, $\nu = 0$.

basic algorithm was extended to take advantage of the additional connectivity offered by the mesh. This yielded the BST broadcast algorithm for meshes. Its predicted performance falls midway between the ST and RH broadcast algorithms for meshes. The ST broadcast algorithm performs best for very short messages and the RH broadcast algorithm performs best for very long messages.

None of these algorithms require knowledge of machine-dependent constants for network latency and bandwidth to obtain good performance. This means that these algorithms will have relatively stable performance as hardware and operating system software changes. While better performance for a specific machine can probably be obtained by designing pipelined algorithms, such as those given in [12] for hypercubes, or by constructing hybrid algorithms, as in [5,9], such algorithms will be more sensitive to changes in machine characteristics than the

algorithms described here.

In Section 4 it was shown that the ST, BST, and RH broadcast algorithms for linear arrays can be extended so that any node, not just node 0, can act as the source of the broadcast message. There is no cost penalty when another node is chosen to be the source. It is conjectured that these results can be extended to meshes as well.

Two techniques for generalizing these broadcast algorithms to linear arrays and meshes of arbitrary size were given in Section 5. The method of companions can be used to generalize all of the broadcast algorithms given here. It was shown that using this method to broadcast on a mesh that does not have rows and columns with powers of 2 nodes approximately doubles the cost of the broadcast. It is likely that campanions can also be used to generalize other global communication algorithms, such as the complete exchange algorithms for meshes given in [5]. A second generalization technique, called virtual nodes, was described that provides slightly better performance than companions for the BST broadcast algorithm for linear arrays with $\nu = 0$.

Finally, the actual performance of the ST, BST and RH broadcast algorithms for linear arrays was measured on the Intel Delta mesh. These measurements generally confirmed the analytic results of Section 3 but showed that a more precise model of the communication network is desirable. Future work will include implementations of these algorithms for meshes, additional study of the communication model for the Delta, and preparation of a model for the communication network of the Intel Paragon mesh.

8. Acknowledgments

This research was performed in part using the Intel Touchstone Delta System operated by the California Institute of Technology on behalf of the Concurrent Supercomputing Consortium. Access to this facility was provided through the Center for Research on Parallel Computing.

9. References

[1] M. Barnett, R. Littlefield, D. G. Payne, and R. van de Geijn. Global combine on mesh architectures with wormhole routing. In *Proceedings of the 7th International Parallel Processing Symposium*. IEEE Computer Society Press, April 1993.

[2] M. Barnett, D. G. Payne, and R. van de Geijn. Optimal minimum spanning tree broadcasting in mesh-connected architectures. Technical Report TR-91-38, Dept. of Computer Sciences, Univ. of Texas, December 1991.

[3] Jean-Claude Bermond, Philippe Michallon, and Denis Trystram. Broadcasting in wraparound meshes with parallel monodirectional links. *Parallel Computing*, 18:639–648, 1992.

[4] Rudolf Berrendorf and Jukka Helin. Evaluating the basic performance of the Intel iPSC/860 parallel computer. *Concurrency: Practice and Experience*, 4(3):223–240, May 1992.

[5] Shahid H. Bokhari. Multiphase complete exchange on a circuit switched hypercube. Technical Report ICASE Report No. 91-5, ICASE, January 1991.

[6] Shahid H. Bokhari and Harry Berryman. Complete exchange on a circuit switched mesh. In *Proceedings of the Scalable High Performance Computing Conference*, pages 300–306, April 1992.

[7] Thomas H. Dunigan. Communication performance of the Intel Touchstone delta mesh. Technical Report ORNL/TM-11983, ORNL, January 1992.

[8] Pierre Fraigniaud, Serge Miguet, and Yves Robert. Scattering on a ring of processors. *Parallel Computing*, 13:377–383, 1990.

[9] Ching-Tien Ho and M. T. Raghunath. Efficient communication primitives on hypercubes. *Concurrency: Practice and Experience*, 4(6):427–457, September 1992.

[10] Roger W. Hockney and Edward A. Carmona. Comparison of communication on the Intel iPSC/860 and Touchstone Delta. *Parallel Computing*, 18:1067–1072, 1992.

[11] Intel Supercomputer Systems Division, Beaverton, Oregon. *A Touchstone DELTA System Description*, February 1991.

[12] S. Lennart Johnsson and Ching-Tien Ho. Spanning graphs for optimum broadcasting and personalized communication in hypercubes. *IEEE Trans. Computers*, 38(9):1249–1268, September 1989.

[13] Rik Littlefield. Characterizing and tuning communications performance on the Touchstone DELTA and iPSC/860. In *Proceedings of the Intel Supercomputer Users' Group 1992 Annual Users' Conference*, pages 309–313, October 4-7 1992.

[14] Youcef Saad and Martin H. Schultz. Data communication in parallel architectures. *Parallel Computing*, 11:131–150, 1989.

[15] Martin Simmen. Comments on broadcast algorithms for two-dimensional grids. *Parallel Computing*, 17:109–112, 1991.

[16] Robert A. van de Geijn. Efficient global combine operations. In *Proceedings of the Sixth Distributed Memory Computing Conference*, pages 291–294. IEEE Computer Society Press, 1991.

INTERNAL DISTRIBUTION

1. B. R. Appleton
2. J. Choi
3. T. S. Darland
4. J. J. Dongarra
5. D. J. Dudziak
6. T. H. Dunigan
7. G. A. Geist
8. M. R. Leuze
9. C. E. Oliver
10. B. W. Peyton

11–15. S. A. Raby

16–20. R. F. Sincovec

21. D. W. Walker

22–26. R. C. Ward

27. P. H. Worley
28. R. W. Brockett (Consultant)
29. J. E. Leiss (Consultant)
30. N. Moray (Consultant)
31. M. F. Wheeler (Consultant)
32. Central Research Library
33. ORNL Patent Office
34. K-25 Applied Technology Library
35. Y-12 Technical Library
36. Laboratory Records Dept. - RC

37–38. Laboratory Records Dept.

EXTERNAL DISTRIBUTION

39. Dan Anderson, Ford Motor Co., Product and Manufacturing Systems, Mail Drop 10, EEC Building, P.O. Box 2053, Dearborn, MI 48121-2053

40. Mike Barnett, Laboratory for Applied Logic, Department of Computer Science, University of Idaho, Moscow, Idaho 83843

41. Dr. Eric Barsczc, Mail Stop T-045, NASA Ames Research Center, Moffett Field, CA 94035

42. Dr. Edward H. Barsis, Computer Science and Mathematics, P. O. Box 5800, Sandia National Laboratory, Albuquerque, NM 87185

43. Scott Berryman, Yale University, Computer Science Department, 51 Prospect Street, New Haven, CT 06520

44. Professor Shahid Bokhari, Dept. of Electrical Engineering, University of Engineering and Technology, Lahore, Pakistan

45. Tom Crockett, Mail Stop 152C, NASA Langley Research Center, Hampton, VA 23665-5225

46. Mark A. Davis, Goddard Space Flight Center, 10210 Greenbelt Rd, Suite 700, Seabrook, MD 20706

47. Professor Larry Dowdy, Computer Science Department, Vanderbilt University, Nashville, TN 37235

48. Professor Geoffrey C. Fox, Northeast Parallel Architectures Center, 111 College Place, Syracuse University, Syracuse, NY 13244-4100

49. Prof. Pierre Fraigniaud, Laboratoire de l'Informatique du Parallelisme - IMAG, Ecole Normale Superieure de Lyon, 69394 Lyon Cedex 07, France

50. Professor Dennis B. Gannon, Computer Science Department, Indiana University, Bloomington, IN 47401

51. Dr. J. Alan George, Vice President, Academic and Provost, Needles Hall, University of Waterloo, Waterloo, Ontario, CANADA N2L 3G1,

52. Dr. Gene H. Golub, Computer Science Department, Stanford University, Stanford, CA 94305

53. Dr. John Gustafson, 236 Wilhelm, Ames Laboratory, Iowa State University, Ames, IA 50011

54. Dr. Dan Hitchcock, Office of Scientific Computing, ER-7, Applied Mathematical Sciences, Office of Energy Research, U. S. Department of Energy, Washington, DC 20585

55. Dr. Ching-Tien Ho, Computer Science, IBM Almaden Research Center, 650 Harry Road, San Jose, CA 95120

56. Dr. Gary Johnson, Office of Scientific Computing, ER-7, Applied Mathematical Sciences, Office of Energy Research, U. S. Department of Energy, Washington, DC 20585

57. Dr. Hans Kaper, Mathematics and Computer Science Division, Argonne National Laboratory, 9700 S. Cass Avenue, Bldg. 221, Argonne, IL 60439

58. Dr. Kenneth Kennedy, Department of Computer Science, Rice University, P. O. Box 1892, Houston, Texas 77001

59. Dr. Tom Kitchens, ER-7, Applied Mathematical Sciences, Office of Scientific Computing, Office of Energy Research, Office G-437, Germantown, Washington, DC 20585

60. Professor S. Lakshmivarahan, School of Electrical Engineering and Computer Science, University of Oklahoma, 202 West Boyd, Room 219, Norman, OK 73019

61. Professor Peter Lax, Courant Institute for Mathematical Sciences, New York University, 251 Mercer Street, New York, NY 10012

62. Dr. John G. Lewis, Boeing Computer Services, P. O. Box 24346, MS 7L-21, Seattle, WA 98124-0346

63. Dr. R. Littlefield, Pacific Northwest Laboratory, P.O. Box 999, Richland, WA 99352

64. Dr. James McGraw, Lawrence Livermore National Laboratory, L-306, P. O. Box 808, Livermore, CA 94550

65. Dr. David Nelson, Director of Office of Scientific Computing, ER-7, Applied Mathematical Sciences, Office of Energy Research, U. S. Department of Energy, Washington, DC 20585

66. Professor James M. Ortega, Department of Applied Mathematics, University of Virginia, Thornton Hall, Charlottesville, VA 22901

67. Dr. David G. Payne, Intel Corporation, Supercomputer Systems Division, 15201 NW Greenbrier Parkway, Beaverton, OR 97006,

68. Dr. Paul Pierce, Intel Corporation, Supercomputer Systems Division, 15201 NW Greenbrier Parkway, Beaverton, OR 97006,

69. Professor Daniel A. Reed, Computer Science Department, University of Illinois, Urbana, IL 61801

70. Professor Ahmed Sameh, University of Illinois at Urbana-Champaign, Center for Supercomputer R&D, 469 CSRL, 1308 West Main St., Urbana, IL 61801

71. Dr. David S. Scott, Intel Scientific Computers, 15201 NW Greenbrier Parkway, Beaverton, OR 97006

72-76. Professor Steven R. Seidel, Department of Computer Science, Michigan Technological University, 1400 Townsend Drive, Houghton, MI 49931-1295

77. Dr. Horst Simon, NASA Ames Research Center, Mail Stop T-045-1, Moffett Field, CA 94035

78. Dr. Paul N. Swartztrauber, National Center for Atmospheric Research, P. O. Box 3000, Boulder, CO 80307

79. Professor Robert van de Geijn, Department of Computer Sciences, The University of Texas at Austin, Austin, TX 78712-1188

80. Dr. Andrew B. White, Los Alamos National Laboratory, P. O. Box 1663, MS-265, Los Alamos, NM 87545

81. Office of Assistant Manager for Energy Research and Development, U.S. Department of Energy, Oak Ridge Operations Office, P. O. Box 2001, Oak Ridge, TN 37831-8600

82-83. Office of Scientific and Technical Information, P. O. Box 62, Oak Ridge, TN 37830

DATE
FILMED
8/4/93
END