tion and the data to be broadcast or reduced contains $m$ words. The broadcast or reduction procedure involves log $p$ point-to-point simple message transfers, each at a time cost of $t_s + t_w m$. Therefore, the total time taken by the procedure is

**Equation 4.1.**

$$T = (t_s + t_w m) \log p.$$

# All-to-All Broadcast and Reduction

All-to-all broadcast is a generalization of one-to-all broadcast in which all $p$ nodes simultaneously initiate a broadcast. A process sends the same $m$-word message to every other process, but different processes may broadcast different messages. All-to-all broadcast is used in matrix operations, including matrix multiplication and matrix-vector multiplication. The dual of all-to-all broadcast is all-to-all reduction, in which every node is the destination of an all-to-one reduction (Problem 4.8). **Figure 4.8** illustrates all-to-all broadcast and all-to-all reduction.

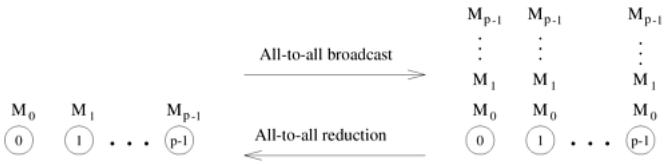


**Figure 4.8. All-to-all broadcast and all-to-all reduction.**

One way to perform an all-to-all broadcast is to perform $p$ one-to-all broadcasts, one starting at each node. If performed naively, on some architectures this approach may take up to $p$ times as long as a one-to-all broadcast. It is possible to use the communication links in the interconnection network more efficiently by performing all $p$ one-to-all broadcasts simultaneously so that all messages traversing the same path at the same time are concatenated into a single message whose size is the sum of the sizes of individual messages.

The following sections describe all-to-all broadcast on linear array, mesh, and hypercube topologies.

## Linear Array and Ring

While performing all-to-all broadcast on a linear array or a ring, all communication links can be kept busy simultaneously until the operation is complete because each node always has some information that it can pass along to its neighbor. Each node first sends to one of its neighbors the data it needs to broadcast. In subsequent steps, it forwards the data received from one of its neighbors to its other neighbor.

Figure 4.9 illustrates all-to-all broadcast for an eight-node ring. The same procedure would also work on a linear array with bidirectional links. As with the previous figures, the integer label of an arrow indicates the time step during which the message is sent. In all-to-all broadcast, $p$ different messages circulate in the $p$-node ensemble. In Figure 4.9, each message is identified by its initial source, whose label appears in parentheses along with the time step. For instance, the arc labeled 2 (7) between nodes 0 and 1 represents the data communicated in time step 2 that node 0 received from node 7 in the preceding step. As Figure 4.9 shows, if communication is performed circularly in a single direction, then each node receives all $(p - 1)$ pieces of information from all other nodes in $(p - 1)$ steps.

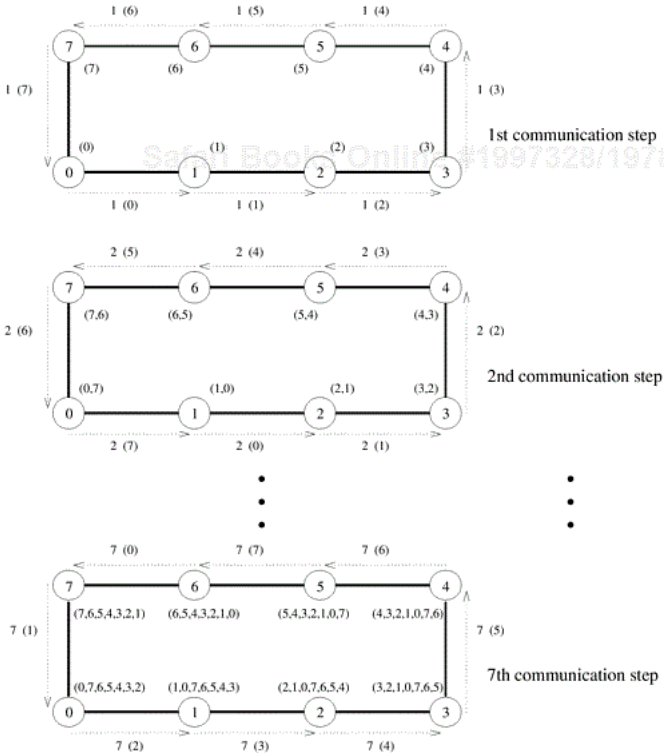


**Figure 4.9. All-to-all broadcast on an eight-node ring. The label of each arrow shows the time step and, within parentheses, the label of the node that owned the current message being transferred before the beginning of the broadcast. The number(s) in parentheses next to each node are the labels of nodes from which data has been received prior to the current communication step. Only the first, second, and last communication steps are shown.**

**Algorithm 4.4** gives a procedure for all-to-all broadcast on a $p$-node ring. The initial message to be broadcast is known locally as *my_msg* at each node. At the end of the procedure, each node stores the collection of all $p$ messages in *result*. As the program shows, all-to-all broadcast on a mesh applies the linear array procedure twice, once along the rows and once along the columns.

**Example 4.4. All-to-all broadcast on a *p*-node ring.**

```
1.   procedure ALL_TO_ALL_BC_RING(my_id, my_msg, p, result)
2.   begin
3.      left := (my_id - 1) mod p;
4.      right := (my_id + 1) mod p;
5.      result := my_msg;
6.      msg := result;
7.      for i := 1 to p - 1 do
8.          send msg to right;
9.          receive msg from left;
10.         result := result ∪ msg;
11.     endfor;
12.  end ALL_TO_ALL_BC_RING
```

In all-to-all reduction, the dual of all-to-all broadcast, each node starts with $p$ messages, each one destined to be accumulated at a distinct node. All-to-all reduction can be performed by reversing the direction and sequence of the messages. For example, the first communication step for all-to-all reduction on an 8-node ring would correspond to the last step of **Figure 4.9** with node 0 sending *msg*[1] to 7 instead of receiving it. The only additional step required is that upon receiving a message, a node must combine it with the local copy of the message that has the same destination as the received message before forwarding the combined message to the next neighbor. **Algorithm 4.5** gives a procedure for all-to-all reduction on a $p$-node ring.

class-room exercise: Run the reduction on the ring

**Example 4.5. All-to-all reduction on a *p*-node ring.**

```
1.   procedure ALL_TO_ALL_RED_RING(my_id, my_msg, p, result)
2.   begin
3.      left := (my_id - 1) mod p;
4.      right := (my_id + 1) mod p;
5.      recv := 0;
6.      for i := 1 to p - 1 do
7.          j := (my_id + i) mod p;
8.          temp := msg[j] + recv; // reduction
9.          send temp to left;
10.         receive recv from right;
11.     endfor;
12.     result := msg[my_id] + recv;
13.  end ALL_TO_ALL_RED_RING
```

## Mesh

Just like one-to-all broadcast, the all-to-all broadcast algorithm for the 2-D mesh is based on the linear array algorithm, treating rows and columns of the mesh as linear arrays. Once again, communication takes place in two phases. In the first phase, each row of the mesh performs an all-to-all broadcast using the procedure for the linear array. In this phase, all nodes collect $\sqrt{p}$ messages corresponding to the $\sqrt{p}$ nodes of their respective rows. Each node consolidates this information into a single message of size $m\sqrt{p}$, and proceeds to the second communication phase of the algorithm. The second communication phase is a columnwise all-to-all broadcast of the consolidated messages. By the end of this phase, each node obtains all $p$ pieces of $m$-word data that originally resided on different nodes. The distribution of data among the nodes of a $3 \times 3$ mesh at the beginning of the first and the second phases of the algorithm is shown in Figure 4.10.

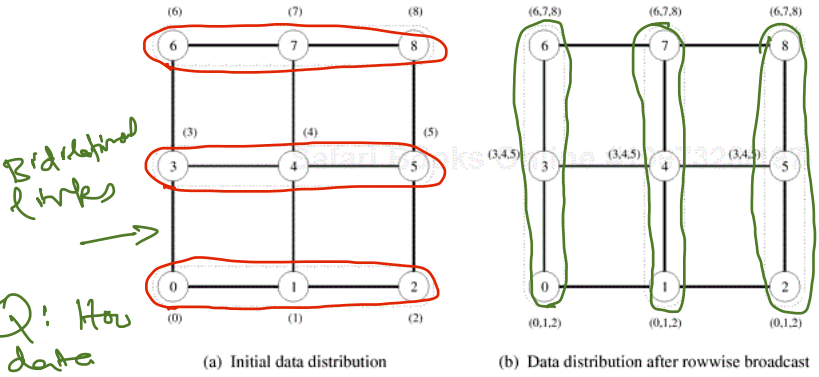


**Figure 4.10. All-to-all broadcast on a 3 × 3 mesh. The groups of nodes communicating with each other in each phase are enclosed by dotted boundaries. By the end of the second phase, all nodes get (0,1,2,3,4,5,6,7) (that is, a message from each node).**

**Algorithm 4.6** gives a procedure for all-to-all broadcast on a $\sqrt{p} \times \sqrt{p}$ mesh. The mesh procedure for all-to-all reduction is left as an exercise for the reader (Problem 4.4).

**Example 4.6. All-to-all broadcast on a square mesh of *p* nodes.**

```
1.   procedure ALL_TO_ALL_BC_MESH(my_id, my_msg, p, result)
2.   begin

/* Communication along rows */
3.       left := my_id - (my_id mod √p) + (my_id - 1)mod √p;
4.       right := my_id - (my_id mod √p) + (my_id + 1) mod √p;
5.       result := my_msg;
6.       msg := result;
7.       for i := 1 to √p - 1 do
8.           send msg to right;
9.           receive msg from left;
10.          result := result ∪ msg;
11.      endfor;

/* Communication along columns */
12.      up := (my_id - √p) mod p;
13.      down := (my_id + √p) mod p;
14.      msg := result;
15.      for i := 1 to √p - 1 do
16.          send msg to down;
17.          receive msg from up;
18.          result := result ∪ msg;
```

```
19.      endfor;
20.  end ALL_TO_ALL_BC_MESH
```

## Hypercube

The hypercube algorithm for all-to-all broadcast is an extension of the mesh algorithm to log $p$ dimensions. The procedure requires log $p$ steps. Communication takes place along a different dimension of the $p$-node hypercube in each step. In every step, pairs of nodes exchange their data and double the size of the message to be transmitted in the next step by concatenating the received message with their current data. **Figure 4.11** shows these steps for an eight-node hypercube with bidirectional communication channels.

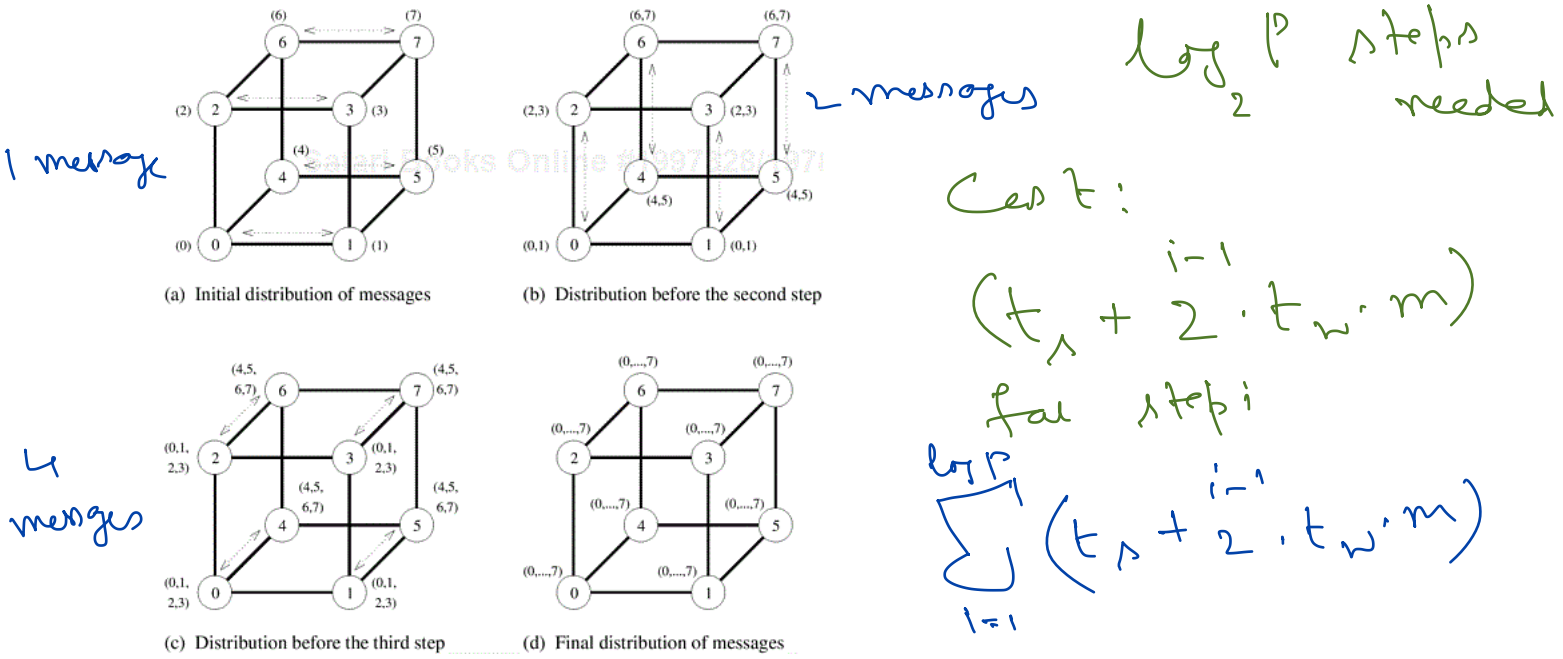


**Figure 4.11. All-to-all broadcast on an eight-node hypercube.**

**Algorithm 4.7** gives a procedure for implementing all-to-all broadcast on a $d$-dimensional hypercube. Communication starts from the lowest dimension of the hypercube and then proceeds along successively higher dimensions (Line 4). In each iteration, nodes communicate in pairs so that the labels of the nodes communicating with each other in the $i$ th iteration differ in the $i$ th least significant bit of their binary representations (Line 5). After an iteration's communication steps, each node concatenates the data it receives during that iteration with its resident data

(Line 8). This concatenated message is transmitted in the following iteration.

**Example 4.7. All-to-all broadcast on a *d*-dimensional hypercube.**

```
1.   procedure ALL_TO_ALL_BC_HCUBE(my_id, my_msg, d, result)
2.   begin
3.      result := my_msg;
4.      for i := 0 to d - 1 do
5.          partner := my id XOR 2^i;
6.          send result to partner;
7.          receive msg from partner;
8.          result := result ∪ msg;
9.      endfor;
10.  end ALL_TO_ALL_BC_HCUBE
```

As usual, the algorithm for all-to-all reduction can be derived by reversing the order and direction of messages in all-to-all broadcast. Furthermore, instead of concatenating the messages, the reduction operation needs to select the appropriate subsets of the buffer to send out and accumulate received messages in each iteration. **Algorithm 4.8** gives a procedure for all-to-all reduction on a *d*-dimensional hypercube. It uses *senloc* to index into the starting location of the outgoing message and *recloc* to index into the location where the incoming message is added in each iteration.

**Example 4.8. All-to-all broadcast on a *d*-dimensional hypercube. AND and XOR are bitwise logical-and and exclusive-or operations, respectively.**

```
1.   procedure ALL_TO_ALL_RED_HCUBE(my_id, msg, d, result)
2.   begin
3.      recloc := 0;
4.      for i := d - 1 to 0 do
5.          partner := my_id XOR 2^i;
6.          j := my_id AND 2^i;
```

```
7.          k := (my_id XOR 2^i) AND 2^i;
8.          senloc := recloc + k;
9.          recloc := recloc + j;
10.         send msg[senloc .. senloc + 2^i - 1] to partner;
11.         receive temp[0 .. 2^i - 1] from partner;
12.         for j := 0 to 2^i - 1 do
13.             msg[recloc + j] := msg[recloc + j] + temp[j];
14.         endfor;
15.     endfor;
16.     result := msg[my_id];
17. end ALL_TO_ALL_RED_HCUBE
```

## Cost Analysis

On a ring or a linear array, all-to-all broadcast involves $p - 1$ steps of communication between nearest neighbors. Each step, involving a message of size $m$, takes time $t_s + t_w m$. Therefore, the time taken by the entire operation is

**Equation 4.2.**

$$T = (t_s + t_w m)(p - 1).$$

Similarly, on a mesh, the first phase of $\sqrt{p}$ simultaneous all-to-all broadcasts (each among $\sqrt{p}$ nodes) concludes in time $(t_s + t_w m)(\sqrt{p} - 1)$. The number of nodes participating in each all-to-all broadcast in the second phase is also $\sqrt{p}$, but the size of each message is now $m\sqrt{p}$. Therefore, this phase takes time $(t_s + t_w m\sqrt{p})(\sqrt{p} - 1)$ to complete. The time for the entire all-to-all broadcast on a $p$-node two-dimensional square mesh is the sum of the times spent in the individual phases, which is

**Equation 4.3.**

$$T = 2t_s(\sqrt{p} - 1) + t_w m(p - 1).$$

On a $p$-node hypercube, the size of each message exchanged in the $i$ th of the log $p$ steps is $2^{i-1}m$. It takes a pair of nodes time $t_s + 2^{i-1}t_w m$ to send and receive messages from each other during the $i$ th step. Hence, the time to complete the entire procedure is

**Equation 4.4.**

Geometric series sum $S_n = \frac{a(r^n - 1)}{r - 1}$

$$T = \sum_{i=1}^{\log p} (t_s + 2^{i-1} t_w m) = t_s \log p + t_w m(p-1).$$

Equations 4.2, 4.3, and 4.4 show that the term associated with $t_w$ in the expressions for the communication time of all-to-all broadcast is $t_w m(p-1)$ for all the architectures. This term also serves as a lower bound for the communication time of all-to-all broadcast for parallel computers on which a node can communicate on only one of its ports at a time. This is because each node receives at least $m(p-1)$ words of data, regardless of the architecture. Thus, for large messages, a highly connected network like a hypercube is no better than a simple ring in performing all-to-all broadcast or all-to-all reduction. In fact, the straightforward all-to-all broadcast algorithm for a simple architecture like a ring has great practical importance. A close look at the algorithm reveals that it is a sequence of $p$ one-to-all broadcasts, each with a different source. These broadcasts are pipelined so that all of them are complete in a total of $p$ nearest-neighbor communication steps. Many parallel algorithms involve a series of one-to-all broadcasts with different sources, often interspersed with some computation. If each one-to-all broadcast is performed using the hypercube algorithm of Section 4.1.3, then $n$ broadcasts would require time $n(t_s + t_w m) \log p$. On the other hand, by pipelining the broadcasts as shown in Figure 4.9, all of them can be performed spending no more than time $(t_s + t_w m)(p-1)$ in communication, provided that the sources of all broadcasts are different and $n \leq p$. In later chapters, we show how such pipelined broadcast improves the performance of some parallel algorithms such as Gaussian elimination (Section 8.3.1), back substitution (Section 8.3.3), and Floyd's algorithm for finding the shortest paths in a graph (Section 10.4.2).

Observation 1

Another noteworthy property of all-to-all broadcast is that, unlike one-to-all broadcast, the hypercube algorithm cannot be applied unaltered to mesh and ring architectures. The reason is that the hypercube procedure for all-to-all broadcast would cause congestion on the communication channels of a smaller-dimensional network with the same number of nodes. For instance, Figure 4.12 shows the result of performing the third step (Figure 4.11(c)) of the hypercube all-to-all broadcast procedure on a

observation 2

ring. One of the links of the ring is traversed by all four messages and would take four times as much time to complete the communication step.

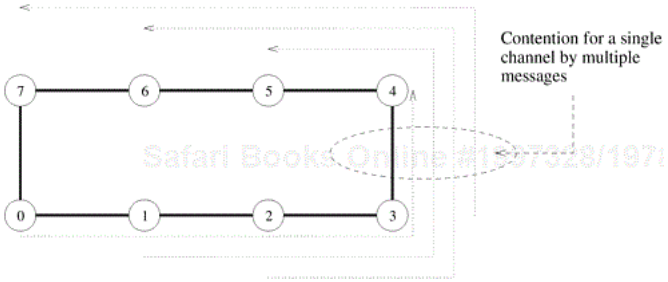


**Figure 4.12. Contention for a channel when the communication step of Figure 4.11(c) for the hypercube is mapped onto a ring.**

# All-Reduce and Prefix-Sum Operations

The communication pattern of all-to-all broadcast can be used to perform some other operations as well. One of these operations is a third variation of reduction, in which each node starts with a buffer of size $m$ and the final results of the operation are identical buffers of size $m$ on each node that are formed by combining the original $p$ buffers using an associative operator. Semantically, this operation, often referred to as the all-reduce operation, is identical to performing an all-to-one reduction followed by a one-to-all broadcast of the result. This operation is different from all-to-all reduction, in which $p$ simultaneous all-to-one reductions take place, each with a different destination for the result.

An all-reduce operation with a single-word message on each node is often used to implement barrier synchronization on a message-passing computer. The semantics of the reduction operation are such that, while executing a parallel program, no node can finish the reduction before each node has contributed a value.

Barrier implementation via all_Reduce

A simple method to perform all-reduce is to perform an all-to-one reduction followed by a one-to-all broadcast. However, there is a faster way to perform all-reduce by using the communication pattern of all-to-all broadcast. Figure 4.11 illustrates this algorithm for an eight-node hypercube. Assume that each integer in parentheses in the figure, instead of denoting a message, denotes a number to be added that originally resided at the node with that integer label. To perform reduction, we follow the