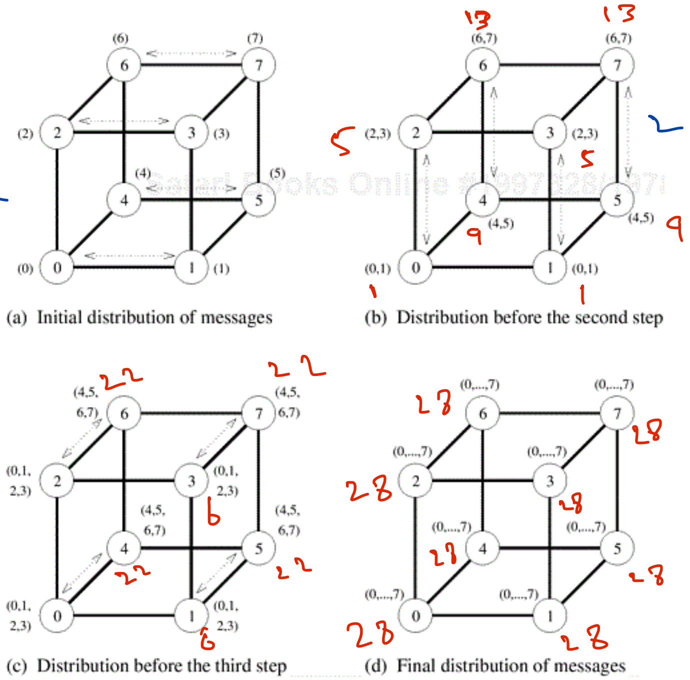
```
        endfor;
    end ALL_TO_ALL_BC_MESH
```

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.



(6)
(7)
6
7
(2)
2
3
(3)
(4)
(5)
4
5
(0)
0
1
(1)
(a) Initial distribution of messages
13
13
(6,7)
(6,7)
5
(2,3)
(2,3)
5
(4,5)
(4,5)
9
9
(0,1)
(0,1)
1
1
(b) Distribution before the second step
(4,5, 6,7)
22
22
(0,1, 2,3)
6
6
6
22
22
6
(c) Distribution before the third step
(0,...,7)
28
(d) Final distribution of messages

1 message

2 messages

4 messages

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

$\log_2 p$ steps needed

Cost:

$(t_s + 2^{i-1} \cdot t_w \cdot m)$

for step i

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

Red annotations for all-reduce section

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   // lowest to highest dimension
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.

Reduction

```
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 $2^i$;
```

```
            $k$ := ($my\_id$ XOR $2^i$) AND $2^i$;
            $senloc$ := $recloc$ + $k$;
            $recloc$ := $recloc$ + $j$;
            send $msg$[$senloc$ .. $senloc + 2^i - 1$] to $partner$;
            receive $temp$[0 .. $2^i - 1$] from $partner$;
            for $j$ := 0 to $2^i - 1$ do
                $msg$[$recloc + j$] := $msg$[$recloc + j$] + temp[$j$];
            endfor;
        endfor;
        $result$ := $msg$[$my\_id$];
    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.

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

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.

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

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: $S_n = \frac{a(r^n - 1)}{r - 1}$

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

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 $n \le p$. 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.

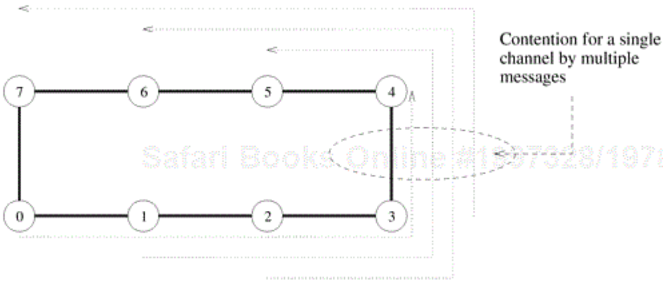


Figure 4.12. **Contention for a channel when the communication step of Figure 4.11(c) for the hypercube is mapped onto a ring.**

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.

all - reduce
=
(1) all-to-one reduction
(2) one-to-all broadcast

$(t_s + m.t_w)$
$\times \log_2 P$
$\times 2$

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.

Example use case
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

See Fig 4.11

communication steps of the all-to-all broadcast procedure, but at the end of each step, add two numbers instead of concatenating two messages. At the termination of the reduction procedure, each node holds the sum (0 + 1 + 2 + ··· + 7) (rather than eight messages numbered from 0 to 7, as in the case of all-to-all broadcast). Unlike all-to-all broadcast, each message transferred in the reduction operation has only one word. The size of the messages does not double in each step because the numbers are added instead of being concatenated. Therefore, the total communication time for all log *p* steps is

Compare with $\textcircled{2} \times (t_s + t_w \cdot m) \log p$
for all-to-one reduction
then one-to-all broadcast

Equation 4.5.

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

Algorithm 4.7 can be used to perform a sum of *p* numbers if *my_msg*, *msg*, and *result* are numbers (rather than messages), and the union operation ('∪') on Line 8 is replaced by addition.

Finding prefix sums (also known as the scan operation) is another important problem that can be solved by using a communication pattern similar to that used in all-to-all broadcast and all-reduce operations. Given *p* numbers $n_0, n_1, \ldots, n_{p-1}$ (one on each node), the problem is to compute the sums $s_k = \Sigma_{i=0}^{k} n_i$ for all *k* between 0 and *p* – 1. For example, if the original sequence of numbers is <3, 1, 4, 0, 2>, then the sequence of prefix sums is <3, 4, 8, 8, 10>. Initially, n_k resides on the node labeled *k*, and at the end of the procedure, the same node holds s_k . Instead of starting with a single numbers, each node could start with a buffer or vector of size *m* and the *m*-word result would be the sum of the corresponding elements of buffers.

prefix sums
problem:

Figure 4.13 illustrates the prefix sums procedure for an eight-node hypercube. This figure is a modification of Figure 4.11. The modification is required to accommodate the fact that in prefix sums the node with label *k* uses information from only the *k*-node subset of those nodes whose labels are less than or equal to *k*. To accumulate the correct prefix sum, every node maintains an additional result buffer. This buffer is denoted by square brackets in Figure 4.13. At the end of a communication step, the content of an incoming message is added to the result buffer only if the message comes from a node with a smaller label than that of the recipient node. The contents of the outgoing message (denoted by parentheses in the figure) are updated with every incoming message, just as in the case of the all-reduce operation. For instance, after the first communication step, nodes 0, 2, and 4 do not add the data received from nodes 1, 3, and 5 to their result buffers. However, the contents of the outgoing messages for the next step are updated.

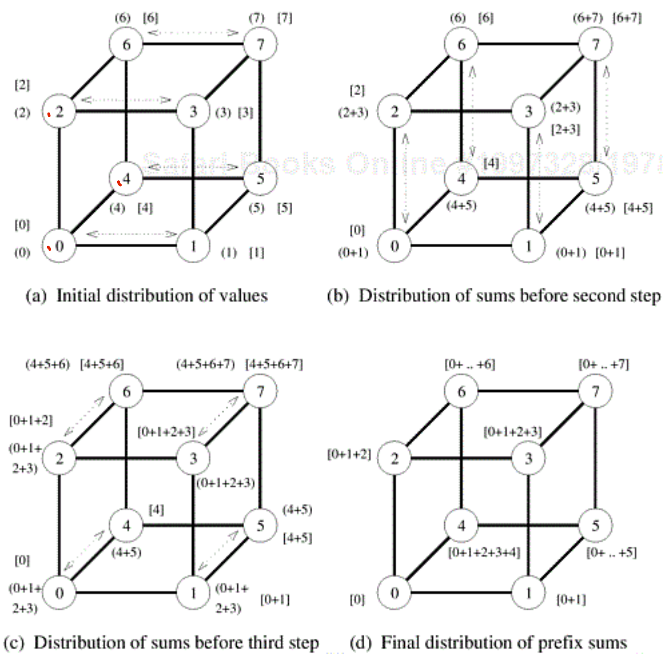
Recall: Cost of all-to-all broadcast/reduce way

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

But now message size does not increase after a time step.

So

$$\text{Cost} = (t_s + m \cdot t_w) \log p$$



(a) Initial distribution of values

(b) Distribution of sums before second step

(c) Distribution of sums before third step

(d) Final distribution of prefix sums

Figure 4.13. Computing prefix sums on an eight-node hypercube. At each node, square brackets show the local prefix sum accumulated in the result buffer and parentheses enclose the contents of the outgoing message buffer for the next step.

Since not all of the messages received by a node contribute to its final result, some of the messages it receives may be redundant. We have omitted these steps of the standard all-to-all broadcast communication pattern from **Figure 4.13**, although the presence or absence of these messages does not affect the results of the algorithm. **Algorithm 4.9** gives a procedure to solve the prefix sums problem on a *d*-dimensional hypercube.

Example 4.9. Prefix sums on a *d*-dimensional hypercube.

```
procedure PREFIX_SUMS_HCUBE(my_id, my number, d, result)
begin
   result := my_number;
   msg := result;
```

```
     for i := 0 to d - 1 do
         partner := my_id XOR $2^i$;
         send msg to partner;
         receive number from partner;
         msg := msg + number;
         if (partner < my_id) then result := result + number;
     endfor;
end PREFIX_SUMS_HCUBE
```

Concatenation

Scatter: one node sends (p-1) messages. Each message is unique.

→ Scatter and Gather

one-to-all personalized comm.

In the scatter operation, a single node sends a unique message of size *m* to every other node. This operation is also known as one-to-all personalized communication. One-to-all personalized communication is different from one-to-all broadcast in that the source node starts with *p* unique messages, one destined for each node. Unlike one-to-all broadcast, one-to-all personalized communication does not involve any duplication of data. The dual of one-to-all personalized communication or the scatter operation is the gather operation, or concatenation, in which a single node collects a unique message from each node. A gather operation is different from an all-to-one reduce operation in that it does not involve any combination or reduction of data. Figure 4.14 illustrates the scatter and gather operations.

Personalized means: a unique message for specific participant

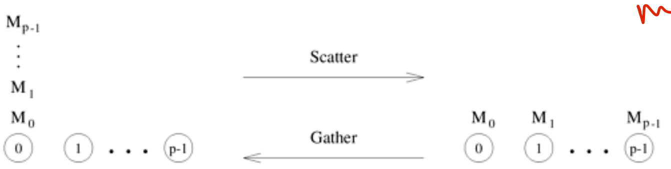
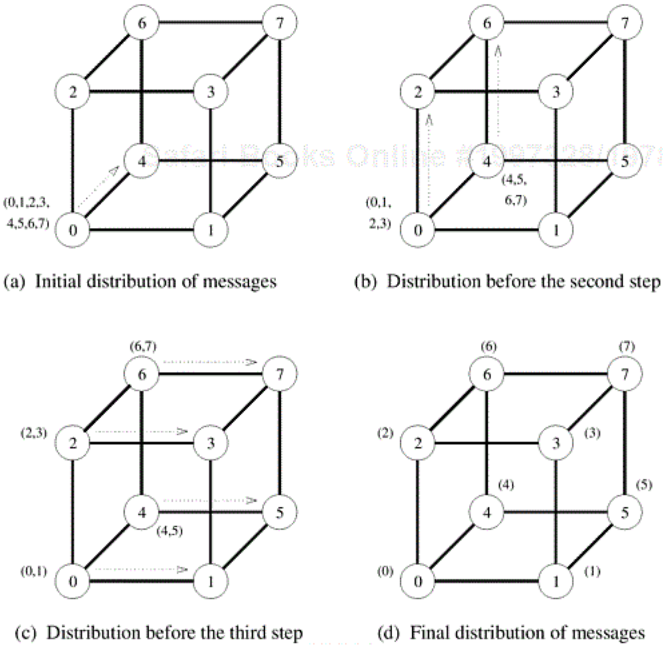


Figure 4.14. Scatter and gather operations.

Q: How gather is different from all-to-one reduction?

Although the scatter operation is semantically different from one-to-all broadcast, the scatter algorithm is quite similar to that of the broadcast. Figure 4.15 shows the communication steps for the scatter operation on an eight-node hypercube. The communication patterns of one-to-all broadcast (Figure 4.6) and scatter (Figure 4.15) are identical. Only the size and the contents of messages are different. In Figure 4.15, the source node (node 0) contains all the messages. The messages are identified by the labels of their destination nodes. In the first communication step, the source transfers half of the messages to one of its neighbors. In subsequent steps, each node that has some data transfers half of it to a neighbor that has yet to receive any data. There is a total of log *p* communication steps corresponding to the log *p* dimensions of the hypercube.



(a) Initial distribution of messages

(b) Distribution before the second step

(c) Distribution before the third step

(d) Final distribution of messages

Figure 4.15. The scatter operation on an eight-node hypercube.

Cost of step i: $t_s + 2^{i-1} . m . t_w$

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

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

Different from one-to-all broadcast because here message sizes are changing (8, to, 4, 2, to 1)

The gather operation is simply the reverse of scatter. Each node starts with an *m* word message. In the first step, every odd numbered node sends its buffer to an even numbered neighbor behind it, which concatenates the received message with its own buffer. Only the even numbered nodes participate in the next communication step which results in nodes with multiples of four labels gathering more data and doubling the sizes of their data. The process continues similarly, until node 0 has gathered the entire data.

Just like one-to-all broadcast and all-to-one reduction, the hypercube algorithms for scatter and gather can be applied unaltered to linear array and mesh interconnection topologies without any increase in the communication time.

Cost Analysis. All links of a *p*-node hypercube along a certain dimension join two *p*/2-node subcubes (Section 2.4.3). As Figure 4.15 illustrates, in each communication step of the scatter operations, data flow from one subcube to another. The data that a node owns before starting communication in a certain dimension are such that half of them need to be sent to a node in the other subcube. In every step, a communicating node keeps half of its data, meant for the nodes in its subcube, and sends the other half to its neighbor in the other subcube. The time in which all data are distributed to their respective destinations is

Equation 4.6.

This portion of the cost is topology independent.

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

The scatter and gather operations can also be performed on a linear array and on a 2-D square mesh in time $t_s \log p + t_w m(p - 1)$ (Problem 4.7). Note that disregarding the term due to message-startup time, the cost of scatter and gather operations for large messages on any *k-d* mesh interconnection network (**Section 2.4.3**) is similar. In the scatter operation, at least $m(p - 1)$ words of data must be transmitted out of the source node, and in the gather operation, at least $m(p - 1)$ words of data must be received by the destination node. Therefore, as in the case of all-to-all broadcast, $t_w m(p - 1)$ is a lower bound on the communication time of scatter and gather operations. This lower bound is independent of the interconnection network.

Observation

All-to-All Personalized Communication

a.k.a.: Total Exchange

In all-to-all personalized communication, each node sends a distinct message of size *m* to every other node. Each node sends different messages to different nodes, unlike all-to-all broadcast, in which each node sends the same message to all other nodes. **Figure 4.16** illustrates the all-to-all personalized communication operation. A careful observation of this figure would reveal that this operation is equivalent to transposing a two-dimensional array of data distributed among *p* processes using one-dimensional array partitioning (**Figure 3.24**). All-to-all personalized communication is also known as total exchange. This operation is used in a variety of parallel algorithms such as fast Fourier transform, matrix transpose, sample sort, and some parallel database join operations.

Example uses

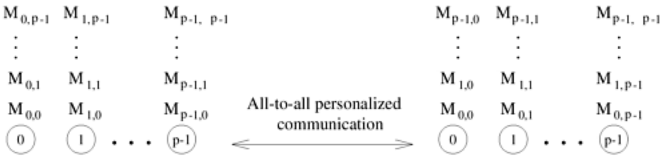


Figure 4.16. All-to-all personalized communication.

EXAMPLE 4.2 MATRIX TRANSPOSITION

The transpose of an $n \times n$ matrix A is a matrix A^T of the same size, such that $A^T[i, j] = A[j, i]$ for $0 \le i, j < n$. Consider an $n \times n$ matrix mapped onto n processors such that each processor contains one full row of the matrix. With this mapping, processor P_i initially contains the elements of the matrix with indices $[i, 0]$, $[i, 1]$, ..., $[i, n - 1]$. After the transposition, element $[i, 0]$ belongs to P_0, element $[i, 1]$ belongs to P_1, and so on. In general, element $[i, j]$ initially resides on P_i , but moves to P_j during the transposition. The data-communication pattern of this procedure is shown in Figure 4.17 for a 4 × 4 matrix mapped onto four processes using one-dimensional rowwise partitioning. Note that in this figure every processor sends a distinct element of the matrix to every other processor. This is an example of all-to-all personalized communication.

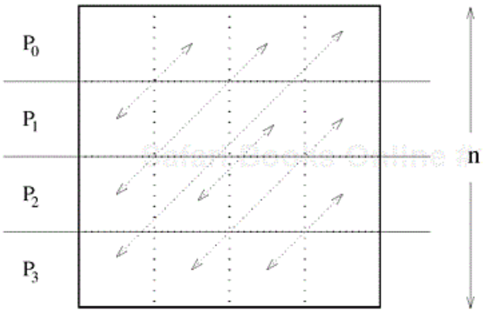


FIGURE 4.17. ALL-TO-ALL PERSONALIZED COMMUNICATION IN TRANSPOSING A 4 × 4 MATRIX USING FOUR PROCESSES.

In general, if we use p processes such that $p \le n$, then each process initially holds n/p rows (that is, n^2/p elements) of the matrix. Performing the transposition now involves an all-to-all personalized communication of matrix blocks of size $n/p \times n/p$, instead of individual elements. ▪

We now discuss the implementation of all-to-all personalized communication on parallel computers with linear array, mesh, and hypercube interconnection networks. The communication patterns of all-to-all personalized communication are identical to those of all-to-all broadcast on all three architectures. Only the size and the contents of messages are different.

Ring

Figure 4.18 shows the steps in an all-to-all personalized communication on a six-node linear array. To perform this operation, every node sends $p - 1$ pieces of data, each of size m. In the figure, these pieces of data are identified by pairs of integers of the form $\{i, j\}$, where i is the source of the message and j is its final destination. First, each node sends all pieces of data as one consolidated message of size $m(p - 1)$ to one of its neighbors (all nodes communicate in the same direction). Of the $m(p - 1)$ words of data received by a node in this step, one m-word packet belongs to it. Therefore, each node extracts the information meant for it from the data received, and forwards the remaining $(p - 2)$ pieces of size m each to the next node. This process continues for $p - 1$ steps. The total size of data being transferred between nodes decreases by m words in each successive step. In every step, each node adds to its collection one m-word packet originating from a different node. Hence, in $p - 1$ steps, every node receives the information from all other nodes in the ensemble.

i = source
j = destination (final)

(P-1) because node won't send the message destined for itself.

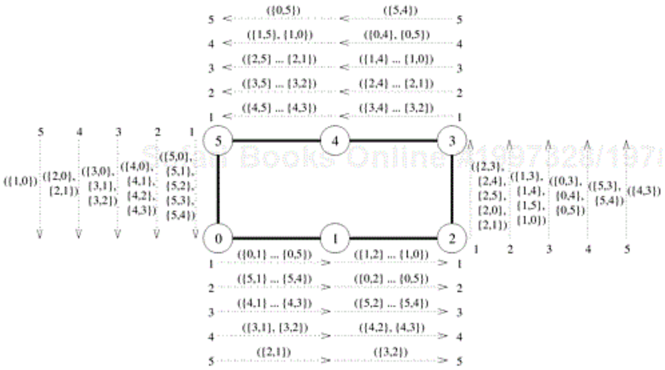


Figure 4.18. All-to-all personalized communication on a six-node ring. The label of each message is of the form $\{x, y\}$, where x is the label of the node that originally owned the message, and y is the label of the node that is the final destination of the message. The label $(\{x_1, y_1\}, \{x_2, y_2\}, ..., \{x_n, y_n\})$ indicates a message that is formed by concatenating n individual messages.

In the above procedure, all messages are sent in the same direction. If half of the messages are sent in one direction and the remaining half are sent in the other direction, then the communication cost due to the t_w can be reduced by a factor of two. For the sake of simplicity, we ignore this constant-factor improvement.

Homework exercise

Cost Analysis. On a ring or a bidirectional linear array, all-to-all personalized communication involves $p - 1$ communication steps. Since the size of the messages transferred in the i th step is $m(p - i)$, the total time taken by this operation is

Equation 4.7.

$$\begin{aligned} T &= \sum_{i=1}^{p-1} (t_s + t_w m(p-i)) \\ &= t_s(p-1) + \sum_{i=1}^{p-1} i t_w m \\ &= (t_s + t_w m p/2)(p-1). \end{aligned}$$

Sum of arithmetic series: $\sum_{i=1}^{n} a_i = \frac{n(a_1 + a_n)}{2}$

or sum of n consecutive no: $1+2+\cdots+n = \frac{n(n+1)}{2}$

In the all-to-all personalized communication procedure described above, each node sends $m(p - 1)$ words of data because it has an m-word packet for every other node. Assume that all messages are sent either clockwise or counterclockwise. The average distance that an m-word packet travels is $(\sum_{i=1}^{p-1} i)/(p-1)$, which is equal to $p/2$. Since there are p nodes, each performing the same type of communication, the total traffic (the total number of data words transferred between directly-connected nodes) on the network is $m(p - 1) \times p/2 \times p$. The total number of inter-node links in the network to share this load is p. Hence, the communication time for this operation is at least $(t_w \times m(p - 1)p^2/2)/p$, which is equal to $t_w m(p - 1)p/2$. Disregarding the message startup time t_s, this is exactly the time taken by the linear array procedure. Therefore, the all-to-all personalized communication algorithm described in this section is optimal.

Mesh

In all-to-all personalized communication on a $\sqrt{p} \times \sqrt{p}$ mesh, each node first groups its p messages according to the columns of their destination nodes. Figure 4.19 shows a 3 × 3 mesh, in which every node initially has nine m-word messages, one meant for each node. Each node assembles its data into three groups of three messages each (in general, $\sqrt{p}$ groups of $\sqrt{p}$