cation 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.

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

This portion of the cost is topology independent.

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 $0 \leq i, j < n$. 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.

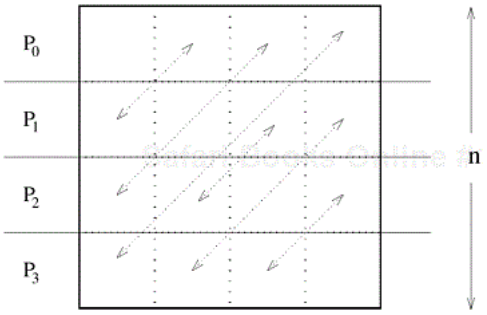


FIGURE 4.17. ALL-TO-ALL PERSONALIZED COMMUNICATION IN TRANSPOSING A 4 × 4 MATRIX USING FOUR PROCESSES.

In general, if we use p processes such that $p \leq n$, 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.

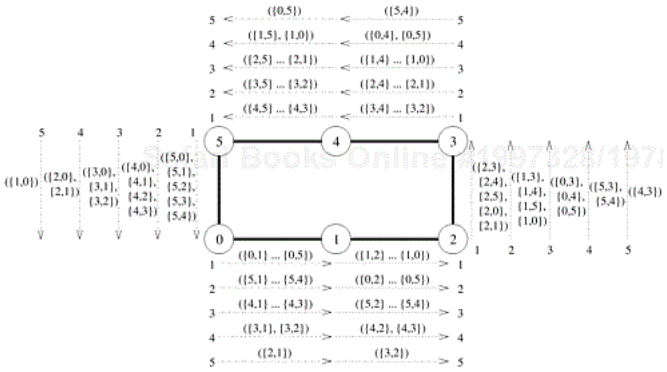


Figure 4.18. All-to-all personalized communication on a six-node ring. The label of each message is of the form $\{x, y\}$, where x is the label of the node that originally owned the message, and y is the label of the node that is the final destination of the message. The label $(\{x_1, y_1\}, \{x_2,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

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.

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

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 $(\Sigma_{i=1}^{p-1} i)/(p-1)$, 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 $\sqrt{p}$

messages each). The first group contains the messages destined for nodes labeled 0, 3, and 6; the second group contains the messages for nodes labeled 1, 4, and 7; and the last group has messages for nodes labeled 2, 5, and 8.

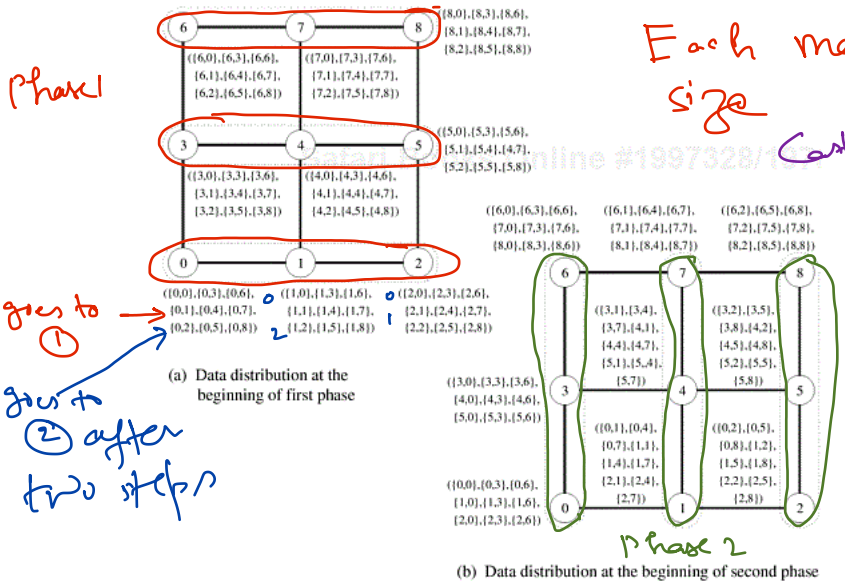


Figure 4.19. The distribution of messages at the beginning of each phase of all-to-all personalized communication on a 3 × 3 mesh. At the end of the second phase, node *i* has messages ({0, *i*}, ..., {8, *i*}), where $0 \leq i \leq 8$. The groups of nodes communicating together in each phase are enclosed in dotted boundaries.

After the messages are grouped, all-to-all personalized communication is performed independently in each row with clustered messages of size $m\sqrt{p}$. One cluster contains the information for all $\sqrt{p}$ nodes of a particular column. **Figure 4.19(b)** shows the distribution of data among the nodes at the end of this phase of communication.

Before the second communication phase, the messages in each node are sorted again, this time according to the rows of their destination nodes; then communication similar to the first phase takes place in all the columns of the mesh. By the end of this phase, each node receives a message from every other node.

Cost Analysis. We can compute the time spent in the first phase by substituting $\sqrt{p}$ for the number of nodes, and $m\sqrt{p}$ for the message size in **Equation 4.7**. The result of this substitution is $(t_s + t_w mp/2)(\sqrt{p} - 1)$. The time spent in the second phase is the same as that in the first phase. Therefore,

the total time for all-to-all personalized communication of messages of size m on a p-node two-dimensional square mesh is

Equation 4.8.

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

The expression for the communication time of all-to-all personalized communication in Equation 4.8 does not take into account the time required for the local rearrangement of data (that is, sorting the messages by rows or columns). Assuming that initially the data is ready for the first communication phase, the second communication phase requires the rearrangement of mp words of data. If t_r is the time to perform a read and a write operation on a single word of data in a node's local memory, then the total time spent in data rearrangement by a node during the entire procedure is $t_r mp$ (Problem 4.21). This time is much smaller than the time spent by each node in communication.

An analysis along the lines of that for the linear array would show that the communication time given by Equation 4.8 for all-to-all personalized communication on a square mesh is optimal within a small constant factor (Problem 4.11).

Hypercube

One way of performing all-to-all personalized communication on a p-node hypercube is to simply extend the two-dimensional mesh algorithm to log p dimensions. Figure 4.20 shows the communication steps required to perform this operation on a three-dimensional hypercube. As shown in the figure, communication takes place in log p steps. Pairs of nodes exchange data in a different dimension in each step. Recall that in a p-node hypercube, a set of $p/2$ links in the same dimension connects two subcubes of $p/2$ nodes each (Section 2.4.3). At any stage in all-to-all personalized communication, every node holds p packets of size m each. While communicating in a particular dimension, every node sends $p/2$ of these packets (consolidated as one message). The destinations of these packets are the nodes of the other subcube connected by the links in current dimension.

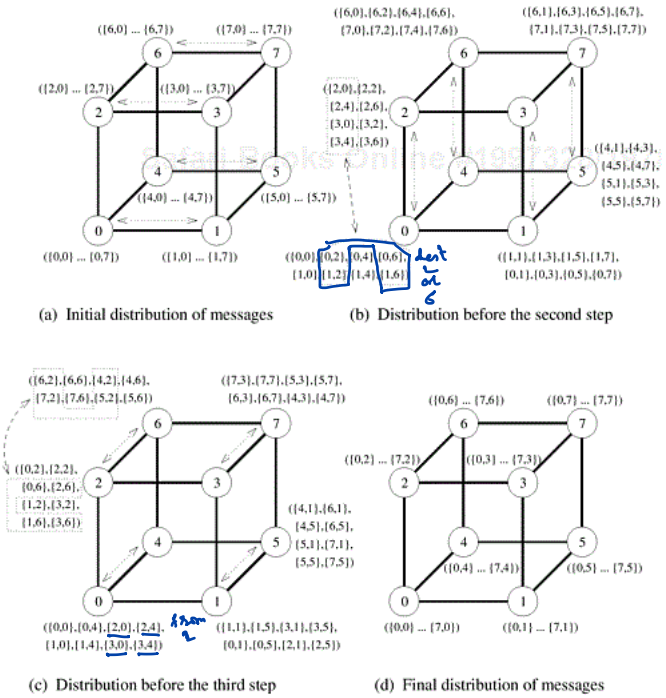


Figure 4.20. An all-to-all personalized communication algorithm on a three-dimensional hypercube.

In the preceding procedure, a node must rearrange its messages locally before each of the log p communication steps. This is necessary to make sure that all $p/2$ messages destined for the same node in a communication step occupy contiguous memory locations so that they can be transmitted as a single consolidated message.

Cost Analysis. In the above hypercube algorithm for all-to-all personalized communication, $mp/2$ words of data are exchanged along the bidirectional channels in each of the log p iterations. The resulting total communication time is

Equation 4.9.

$$T = (t_s + t_w mp/2) \log p.$$

Before each of the log p communication steps, a node rearranges mp words of data. Hence, a total time of $t_r mp \log p$ is spent by each node in local rearrangement of data during the entire procedure. Here t_r is the time needed to perform a read and a write operation on a single word of data in a node's local memory. For most practical computers, t_r is much smaller than t_w; hence, the time to perform an all-to-all personalized communication is dominated by the communication time.

Interestingly, unlike the linear array and mesh algorithms described in this section, the hypercube algorithm is not optimal. Each of the p nodes sends and receives $m(p-1)$ words of data and the average distance between any two nodes on a hypercube is $(\log p)/2$. Therefore, the total data traffic on the network is $p \times m(p-1) \times (\log p)/2$. Since there is a total of $(p \log p)/2$ links in the hypercube network, the lower bound on the all-to-all personalized communication time is

$$\begin{aligned} T &= \frac{t_w p m(p-1)(\log p)/2}{(p \log p)/2} \\ &= t_w m(p-1). \end{aligned}$$

An Optimal Algorithm

An all-to-all personalized communication effectively results in all pairs of nodes exchanging some data. On a hypercube, the best way to perform this exchange is to have every pair of nodes communicate directly with each other. Thus, each node simply performs $p-1$ communication steps, exchanging m words of data with a different node in every step. A node must choose its communication partner in each step so that the hypercube links do not suffer congestion. **Figure 4.21** shows one such congestion-free schedule for pairwise exchange of data in a three-dimensional hypercube. As the figure shows, in the j th communication step, node i exchanges data with node (i XOR j). For example, in part (a) of the figure (step 1), the labels of communicating partners differ in the least significant bit. In part (g) (step 7), the labels of communicating partners differ in all the bits, as the binary representation of seven is 111. In this figure, all the paths in every communication step are congestion-free, and none of the bidirectional links carry more than one message in the same direction. This is true in general for a hypercube of any dimension. If the messages are routed appropriately, a congestion-free schedule exists for the $p-1$ communication steps of all-to-all personalized communication on a p-node hypercube. Recall from **Section 2.4.3** that a message traveling from node i to node j on a hypercube must pass through at least l links, where l is the Hamming distance between i and j (that is, the number of nonzero bits in the binary representation of (i XOR j)). A message traveling from node i to node j traverses links in l dimensions (corresponding to the nonzero bits in the binary representation of (i XOR j)). Although the message

can follow one of the several paths of length l that exist between i and j (assuming $l > 1$), a distinct path is obtained by sorting the dimensions along which the message travels in ascending order. According to this strategy, the first link is chosen in the dimension corresponding to the least significant nonzero bit of (i XOR j), and so on. This routing scheme is known as E-cube routing.

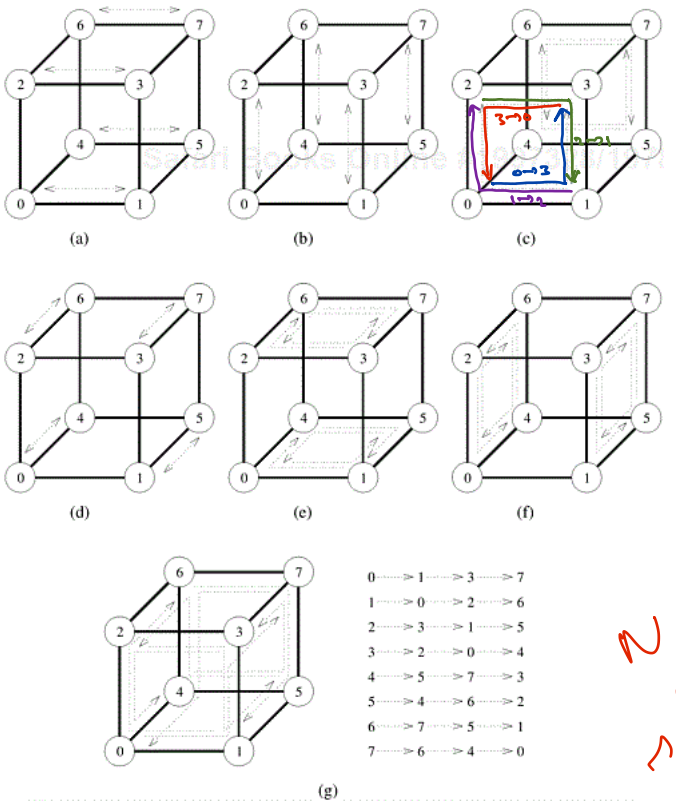


Figure 4.21. Seven steps in all-to-all personalized communication on an eight-node hypercube.

Algorithm 4.10 for all-to-all personalized communication on a d-dimensional hypercube is based on this strategy.

Example 4.10. A procedure to perform all-to-all personalized communication on a d-dimensional hypercube. The message $M_{i,j}$ initially resides on node i and is destined for node j.

```
1.   procedure ALL_TO_ALL_PERSONAL(d, my_id)
2.   begin
3.      for i := 1 to 2^d - 1 do
4.      begin
5.         partner := my_id XOR i;
```

```
6.          send M_my_id, partner to partner;
7.          receive M_partner,my_id from partner;
8.       endfor;
9.    end ALL_TO_ALL_PERSONAL
```

Cost Analysis. E-cube routing ensures that by choosing communication pairs according to **Algorithm 4.10**, a communication time of $t_s + t_w m$ is guaranteed for a message transfer between node i and node j because there is no contention with any other message traveling in the same direction along the link between nodes i and j. The total communication time for the entire operation is

Equation 4.10.

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

A comparison of Equations **4.9** and **4.10** shows the term associated with t_s is higher for the second hypercube algorithm, while the term associated with t_w is higher for the first algorithm. Therefore, for small messages, the startup time may dominate, and the first algorithm may still be useful.

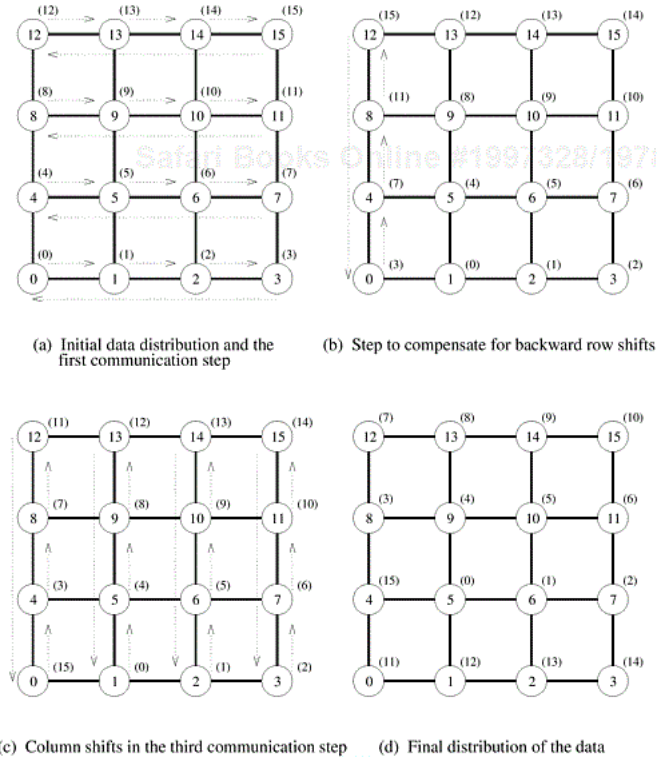
Circular Shift

Circular shift is a member of a broader class of global communication operations known as permutation. A permutation is a simultaneous, one-to-one data redistribution operation in which each node sends a packet of m words to a unique node. We define a circular q-shift as the operation in which node i sends a data packet to node $(i + q) \bmod p$ in a p-node ensemble $(0 < q < p)$. The shift operation finds application in some matrix computations and in string and image pattern matching.

Mesh

The implementation of a circular q-shift is fairly intuitive on a ring or a bidirectional linear array. It can be performed by $\min\{q, p - q\}$ neighbor-to-neighbor communications in one direction. Mesh algorithms for circular shift can be derived by using the ring algorithm.

If the nodes of the mesh have row-major labels, a circular q-shift can be performed on a p-node square wraparound mesh in two stages. This is illustrated in **Figure 4.22** for a circular 5-shift on a 4 × 4 mesh. First, the entire set of data is shifted simultaneously by (q mod $\sqrt{p}$) steps along the rows. Then it is shifted by $\lfloor q/\sqrt{p} \rfloor$ steps along the columns. During the circular row shifts, some of the data traverse the wraparound connection from the highest to the lowest labeled nodes of the rows. All such data packets must shift an additional step forward along the columns to compensate for the $\sqrt{p}$ distance that they lost while traversing the backward edge in their respective rows. For example, the 5-shift in **Figure 4.22** requires one row shift, a compensatory column shift, and finally one column shift.



(a) Initial data distribution and the first communication step

(b) Step to compensate for backward row shifts

(c) Column shifts in the third communication step

(d) Final distribution of the data

Figure 4.22. The communication steps in a circular 5-shift on a 4 × 4 mesh.

In practice, we can choose the direction of the shifts in both the rows and the columns to minimize the number of steps in a circular shift. For instance, a 3-shift on a 4 × 4 mesh can be performed by a single backward row shift. Using this strategy, the number of unit shifts in a direction cannot exceed $\lfloor \sqrt{p}/2 \rfloor$.

Cost Analysis. Taking into account the compensating column shift for some packets, the total time for any circular q-shift on a p-node mesh us-

ing packets of size m has an upper bound of

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

Hypercube

In developing a hypercube algorithm for the shift operation, we map a linear array with 2^d nodes onto a d-dimensional hypercube. We do this by assigning node i of the linear array to node j of the hypercube such that j is the d-bit binary reflected Gray code (RGC) of i. **Figure 4.23** illustrates this mapping for eight nodes. A property of this mapping is that any two nodes at a distance of 2^i on the linear array are separated by exactly two links on the hypercube. An exception is $i = 0$ (that is, directly-connected nodes on the linear array) when only one hypercube link separates the two nodes.

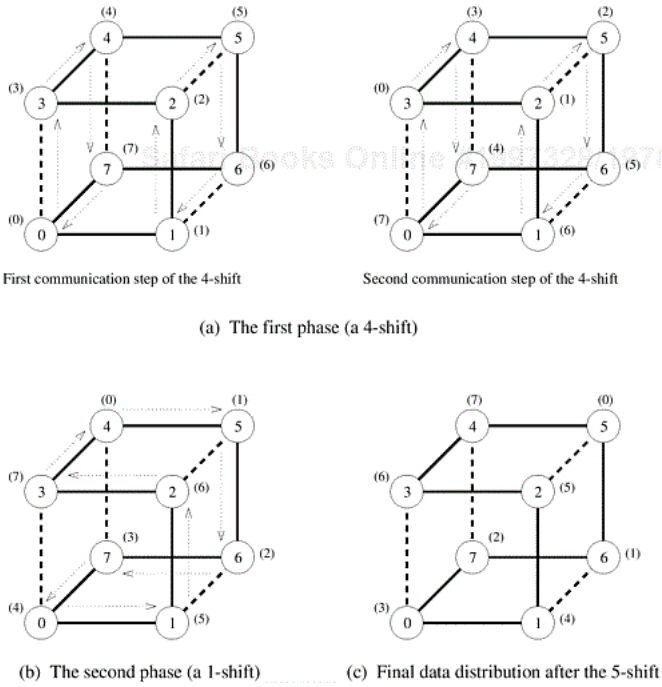


Figure 4.23. The mapping of an eight-node linear array onto a three-dimensional hypercube to perform a circular 5-shift as a combination of a 4-shift and a 1-shift.

To perform a q-shift, we expand q as a sum of distinct powers of 2. The number of terms in the sum is the same as the number of ones in the binary representation of q. For example, the number 5 can be expressed as $2^2 + 2^0$. These two terms correspond to bit positions 0 and 2 in the binary representation of 5, which is 101. If q is the sum of s distinct powers of 2, then the circular q-shift on a hypercube is performed in s phases.

In each phase of communication, all data packets move closer to their respective destinations by short cutting the linear array (mapped onto the hypercube) in leaps of the powers of 2. For example, as **Figure 4.23** shows, a 5-shift is performed by a 4-shift followed by a 1-shift. The number of communication phases in a q-shift is exactly equal to the number of ones in the binary representation of q. Each phase consists of two communication steps, except the 1-shift, which, if required (that is, if the least significant bit of q is 1), consists of a single step. For example, in a 5-shift, the first phase of a 4-shift (**Figure 4.23(a)**) consists of two steps and the second phase of a 1-shift (**Figure 4.23(b)**) consists of one step. Thus, the total number of steps for any q in a p-node hypercube is at most $2 \log p - 1$.

All communications in a given time step are congestion-free. This is ensured by the property of the linear array mapping that all nodes whose mutual distance on the linear array is a power of 2 are arranged in disjoint subarrays on the hypercube. Thus, all nodes can freely communicate in a circular fashion in their respective subarrays. This is shown in **Figure 4.23(a)**, in which nodes labeled 0, 3, 4, and 7 form one subarray and nodes labeled 1, 2, 5, and 6 form another subarray.

The upper bound on the total communication time for any shift of m-word packets on a p-node hypercube is

Equation 4.11.

$$T = (t_s + t_w m)(2 \log p - 1).$$

We can reduce this upper bound to $(t_s + t_w m) \log p$ by performing both forward and backward shifts. For example, on eight nodes, a 6-shift can be performed by a single backward 2-shift instead of a forward 4-shift followed by a forward 2-shift.

We now show that if the E-cube routing introduced in **Section 4.5** is used, then the time for circular shift on a hypercube can be improved by almost a factor of $\log p$ for large messages. This is because with E-cube routing, each pair of nodes with a constant distance l ($i \leq l < p$) has a congestion-free path (Problem 4.22) in a p-node hypercube with bidirectional

channels. **Figure 4.24** illustrates the non-conflicting paths of all the messages in circular q -shift operations for $1 \leq q < 8$ on an eight-node hypercube. In a circular q-shift on a p-node hypercube, the longest path contains $\log p - \gamma(q)$ links, where $\gamma(q)$ is the highest integer j such that q is divisible by 2^j (Problem 4.23). Thus, the total communication time for messages of length m is

Equation 4.12.

$$T = t_s + t_w m.$$

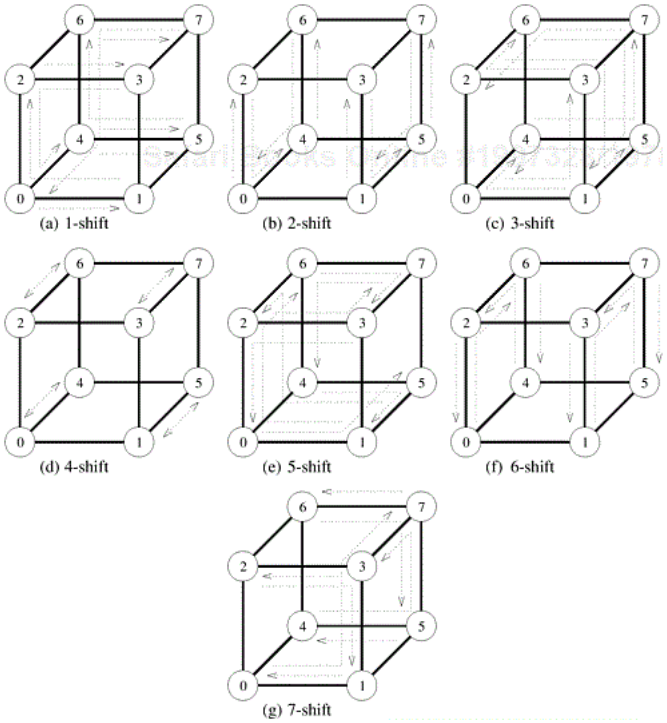


Figure 4.24. Circular q-shifts on an 8-node hypercube for $1 \leq q < 8$.

Improving the Speed of Some Communication Operations

So far in this chapter, we have derived procedures for various communication operations and their communication times under the assumptions that the original messages could not be split into smaller parts and that each node had a single port for sending and receiving data. In this section, we briefly discuss the impact of relaxing these assumptions on some of the communication operations.

Splitting and Routing Messages in Parts

In the procedures described in Sections 4.1–4.6, we assumed that an entire m-word packet of data travels between the source and the destination nodes along the same path. If we split large messages into smaller parts and then route these parts through different paths, we can sometimes utilize the communication network better. We have already shown that, with a few exceptions like one-to-all broadcast, all-to-one reduction, all-reduce, etc., the communication operations discussed in this chapter are asymptotically optimal for large messages; that is, the terms associated with t_w in the costs of these operations cannot be reduced asymptotically. In this section, we present asymptotically optimal algorithms for three global communication operations.

assumption
large messages

Note that the algorithms of this section rely on m being large enough to be split into p roughly equal parts. Therefore, the earlier algorithms are still useful for shorter messages. A comparison of the cost of the algorithms in this section with those presented earlier in this chapter for the same operations would reveal that the term associated with t_s increases and the term associated with t_w decreases when the messages are split. Therefore, depending on the actual values of t_s, t_w, and p, there is a cut-off value for the message size m and only the messages longer than the cut-off would benefit from the algorithms in this section.

One-to-All Broadcast

Consider broadcasting a single message M of size m from one source node to all the nodes in a p-node ensemble. If m is large enough so that M can be split into p parts $M_0, M_1, \ldots, M_{p-1}$ of size m/p each, then a scatter operation (Section 4.4) can place M_i on node i in time $t_s \log p + t_w(m/p)(p-1)$. Note that the desired result of the one-to-all broadcast is to place $M = M_0 \cup M_1 \cup \cdots \cup M_{p-1}$ on all nodes. This can be accomplished by an all-to-all broadcast of the messages of size m/p residing on each node after the scatter operation. This all-to-all broadcast can be completed in time $t_s \log p + t_w(m/p)(p-1)$ on a hypercube. Thus, on a hypercube, one-to-all broadcast can be performed in time

1) Scatter
2) all-to-all broadcast

Equation 4.13.

$$T = 2 \times (t_s \log p + t_w (p-1)\frac{m}{p}) \approx 2 \times (t_s \log p + t_w m).$$

Compared to **Equation 4.1**, this algorithm has double the startup cost, but the cost due to the t_w term has been reduced by a factor of (log p)/2. Similarly, one-to-all broadcast can be improved on linear array and mesh interconnection networks as well.

All-to-One Reduction

All-to-one reduction is a dual of one-to-all broadcast. Therefore, an algorithm for all-to-one reduction can be obtained by reversing the direction and the sequence of communication in one-to-all broadcast. We showed above how an optimal one-to-all broadcast algorithm can be obtained by performing a scatter operation followed by an all-to-all broadcast. Therefore, using the notion of duality, we should be able to perform an all-to-one reduction by performing all-to-all reduction (dual of all-to-all broadcast) followed by a gather operation (dual of scatter). We leave the details of such an algorithm as an exercise for the reader (Problem 4.17).

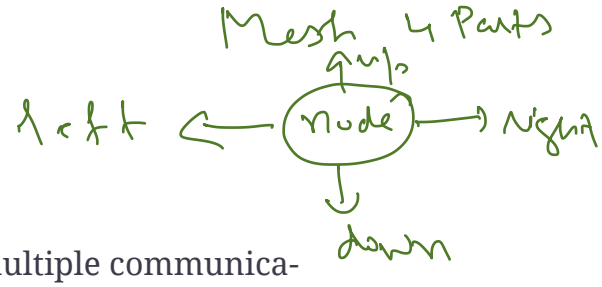
All-Reduce

Since an all-reduce operation is semantically equivalent to an all-to-one reduction followed by a one-to-all broadcast, the asymptotically optimal algorithms for these two operations presented above can be used to construct a similar algorithm for the all-reduce operation. Breaking all-to-one reduction and one-to-all broadcast into their component operations, it can be shown that an all-reduce operation can be accomplished by an all-to-all reduction followed by a gather followed by a scatter followed by an all-to-all broadcast. Since the intermediate gather and scatter would simply nullify each other's effect, all-reduce just requires an all-to-all reduction and an all-to-all broadcast. First, the m-word messages on each of the p nodes are logically split into p components of size roughly m/p words. Then, an all-to-all reduction combines all the i th components on p_i. After this step, each node is left with a distinct m/p-word component of the final result. An all-to-all broadcast can construct the concatenation of these components on each node.

A p-node hypercube interconnection network allows all-to-one reduction and one-to-all broadcast involving messages of size m/p in time $t_s \log p + t_w(m/p)(p-1)$ each. Therefore, the all-reduce operation can be completed in time

Equation 4.14.

$$T = 2 \times (t_s \log p + t_w(p-1)\frac{m}{p}) \approx 2 \times (t_s \log p + t_w m).$$



All-Port Communication

In a parallel architecture, a single node may have multiple communication ports with links to other nodes in the ensemble. For example, each node in a two-dimensional wraparound mesh has four ports, and each node in a d-dimensional hypercube has d ports. In this book, we generally assume what is known as the single-port communication model. In single-port communication, a node can send data on only one of its ports at a time. Similarly, a node can receive data on only one port at a time. However, a node can send and receive data simultaneously, either on the same port or on separate ports. In contrast to the single-port model, an all-port communication model permits simultaneous communication on all the channels connected to a node.

On a p-node hypercube with all-port communication, the coefficients of t_w in the expressions for the communication times of one-to-all and all-to-all broadcast and personalized communication are all smaller than their single-port counterparts by a factor of $\log p$. Since the number of channels per node for a linear array or a mesh is constant, all-port communication does not provide any asymptotic improvement in communication time on these architectures.

Despite the apparent speedup, the all-port communication model has certain limitations. For instance, not only is it difficult to program, but it requires that the messages are large enough to be split efficiently among different channels. In several parallel algorithms, an increase in the size of messages means a corresponding increase in the granularity of computation at the nodes. When the nodes are working with large data sets, the

internode communication time is dominated by the computation time if the computational complexity of the algorithm is higher than the communication complexity. For example, in the case of matrix multiplication, there are n^3 computations for n^2 words of data transferred among the nodes. If the communication time is a small fraction of the total parallel run time, then improving the communication by using sophisticated techniques is not very advantageous in terms of the overall run time of the parallel algorithm.

Example: why all-port solution can be useful

Comm Cost vs Computation Cost

Another limitation of all-port communication is that it can be effective only if data can be fetched and stored in memory at a rate sufficient to sustain all the parallel communication. For example, to utilize all-port communication effectively on a p-node hypercube, the memory bandwidth must be greater than the communication bandwidth of a single channel by a factor of at least $\log p$; that is, the memory bandwidth must increase with the number of nodes to support simultaneous communication on all ports. Some modern parallel computers, like the IBM SP, have a very natural solution for this problem. Each node of the distributed-memory parallel computer is a NUMA shared-memory multiprocessor. Multiple ports are then served by separate memory banks and full memory and communication bandwidth can be utilized if the buffers for sending and receiving data are placed appropriately across different memory banks.

⊖ More mem
⊖ More mem bandwidth

SP = Scalable power

Summary

Table 4.1 summarizes the communication times for various collective communications operations discussed in this chapter. The time for one-to-all broadcast, all-to-one reduction, and the all-reduce operations is the minimum of two expressions. This is because, depending on the message size m, either the algorithms described in Sections **4.1** and **4.3** or the ones described in **Section 4.7** are faster. **Table 4.1** assumes that the algorithm most suitable for the given message size is chosen. The communication-time expressions in **Table 4.1** have been derived in the earlier sections of this chapter in the context of a hypercube interconnection network with cut-through routing. However, these expressions and the corresponding

algorithms are valid for any architecture with a Θ(p) cross-section bandwidth (**Section 2.4.4**). In fact, the terms associated with t_w for the expressions for all operations listed in **Table 4.1**, except all-to-all personalized communication and circular shift, would remain unchanged even on ring and mesh networks (or any *k-d* mesh network) provided that the logical processes are mapped onto the physical nodes of the network appropriately. The last column of **Table 4.1** gives the asymptotic cross-section bandwidth required to perform an operation in the time given by the second column of the table, assuming an optimal mapping of processes to nodes. For large messages, only all-to-all personalized communication and circular shift require the full Θ(p) cross-section bandwidth. Therefore, as discussed in **Section 2.5.1**, when applying the expressions for the time of these operations on a network with a smaller cross-section bandwidth, the t_w term must reflect the effective bandwidth. For example, the bisection width of a p-node square mesh is $\Theta(\sqrt{p})$ and that of a p-node ring is Θ(1). Therefore, while performing all-to-all personalized communication on a square mesh, the effective per-word transfer time would be $\Theta(\sqrt{p})$ times the t_w of individual links, and on a ring, it would be $\Theta(p)$ times the t_w of individual links.

Table 4.1. Summary of communication times of various operations discussed in Sections 4.1–4.7 on a hypercube interconnection network. The message size for each operation is m and the number of nodes is p.

Operation	Hypercube Time	B/W Requirement
One-to-all broadcast, All-to-one reduction	$\min((t_s + t_w m) \log p, 2(t_s \log p + t_w m))$	Θ(1)
All-to-all broadcast,	$t_s \log p + t_w m(p - 1)$	Θ(1)

Operation	Hypercube Time	B/W Requirement
All-to-all reduction		
All-reduce	$\min((t_s + t_w m) \log p, 2(t_s \log p + t_w m))$	$\Theta(1)$
Scatter, Gather	$t_s \log p + t_w m(p - 1)$	$\Theta(1)$
All-to-all personalized	$(t_s + t_w m)(p - 1)$	$\Theta(p)$
Circular shift	$t_s + t_w m$	$\Theta(p)$

Table 4.2. MPI names of the various operations discussed in this chapter.

Operation	MPI Name
One-to-all broadcast	MPI_Bcast
All-to-one reduction	MPI_Reduce
All-to-all broadcast	MPI_Allgather
All-to-all reduction	MPI_Reduce_scatter
All-reduce	MPI_Allreduce
Gather	MPI_Gather
Scatter	MPI_Scatter

Operation	MPI Name
All-to-all personalized	MPI_Alltoall

The collective communications operations discussed in this chapter occur frequently in many parallel algorithms. In order to facilitate speedy and portable design of efficient parallel programs, most parallel computer vendors provide pre-packaged software for performing these collective communications operations. The most commonly used standard API for these operations is known as the Message Passing Interface, or MPI. **Table 4.2** gives the names of the MPI functions that correspond to the communications operations described in this chapter.

Bibliographic Remarks

In this chapter, we studied a variety of data communication operations for the linear array, mesh, and hypercube interconnection topologies. Saad and Schultz [**SS89b**] discuss implementation issues for these operations on these and other architectures, such as shared-memory and a switch or bus interconnect. Most parallel computer vendors provide standard APIs for inter-process communications via message-passing. Two of the most common APIs are the message passing interface (MPI) [**SOHL+96**] and the parallel virtual machine (PVM) [**GBD+94**].

The hypercube algorithm for a certain communication operation is often the best algorithm for other less-connected architectures too, if they support cut-through routing. Due to the versatility of the hypercube architecture and the wide applicability of its algorithms, extensive work has been done on implementing various communication operations on hypercubes [**BOS+91**, **BR90**, **BT97**, **FF86**, **JH89**, **Joh90**, **MdV87**, **RS90b**, **SS89a**, **SW87**]. The properties of a hypercube network that are used in deriving the algorithms for various communication operations on it are described by Saad and Schultz [**SS88**].

The all-to-all personalized communication problem in particular has been analyzed for the hypercube architecture by Boppana and