

# **Basic Communication Operations**

**Ananth Grama, Anshul Gupta,  
George Karypis, and Vipin Kumar**

Based on the text ``Introduction to Parallel Computing'', Addison Wesley, 2003

# Topic Overview

- One-to-All Broadcast and All-to-One Reduction
- All-to-All Broadcast and Reduction
- All-Reduce and Prefix-Sum Operations
- Scatter and Gather
- All-to-All Personalized Communication
- Circular Shift

# Communication Operations in MPI

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
All-to-all personalized	MPI_Alltoall

# Basic Communication Operations: Introduction

- Many interactions in practical parallel programs occur in well-defined patterns involving groups of processors.
- Efficient implementations of these operations can improve performance, reduce development effort and cost, and improve software quality.
- Efficient implementations must leverage underlying architecture.
- We select a descriptive set of architectures to illustrate the process of algorithm design.
  - Ring (linear array), two-dimensional Mesh, Hypercube

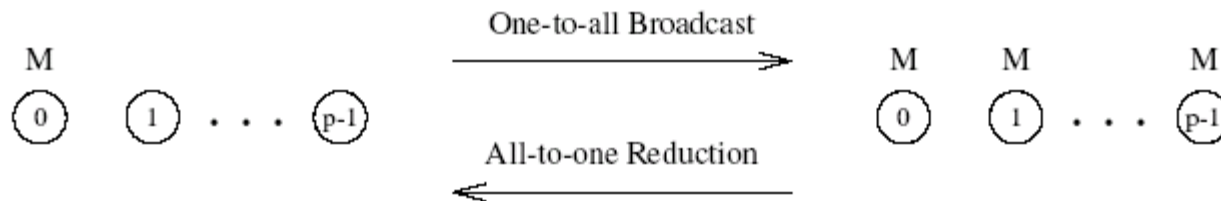
# Basic Communication Operations: Introduction

- Group communication operations are built using point-to-point messaging primitives.
- Communicating a message of size  $m$  over an uncongested network takes time  $t_s + t_w * m$ .
- Where necessary, we take congestion into account explicitly by scaling the  $t_w$  term.
- We assume that the network is bidirectional and that communication is single-ported.

# One-to-All Broadcast and All-to-One Reduction

- One processor has a piece of data (of size  $m$ ) it needs to send to everyone.
- The dual of one-to-all broadcast is *all-to-one reduction*.
- In all-to-one reduction, each processor has  $m$  units of data. These data items must be combined piece-wise (using some associative operator, such as addition or min), and the result made available at a target processor.

# One-to-All Broadcast and All-to-One Reduction



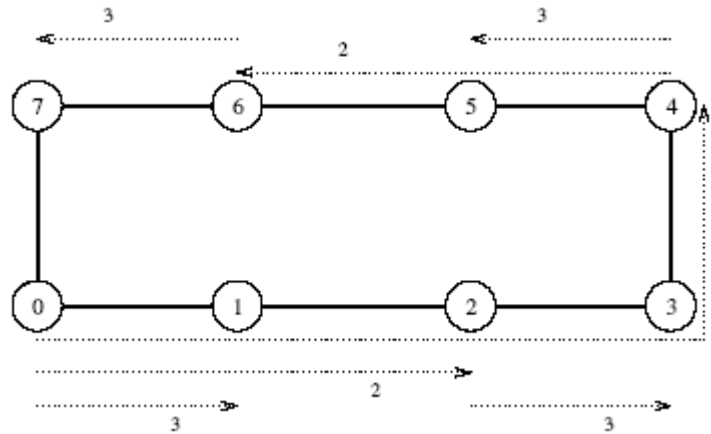
One-to-all broadcast and all-to-one reduction among  $p$  processors.

# One-to-All Broadcast and All-to-One Reduction on Rings

- Simplest way is to send  $p-1$  messages from the source to the other  $p-1$  processors - this is not very efficient.
- Use recursive doubling: source sends a message to a selected processor. We now have two independent problems each on one half of the machines.
- Reduction can be performed by inverting the broadcasting process.

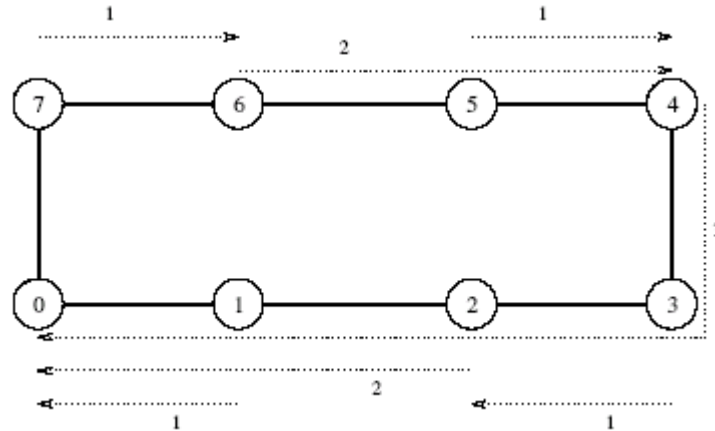


# One-to-All Broadcast



One-to-all broadcast on an eight-node ring. Node 0 is the source of the broadcast. Each message transfer step is shown by a numbered, dotted arrow from the source of the message to its destination. The number on an arrow indicates the time step during which the message is transferred.

# All-to-One Reduction



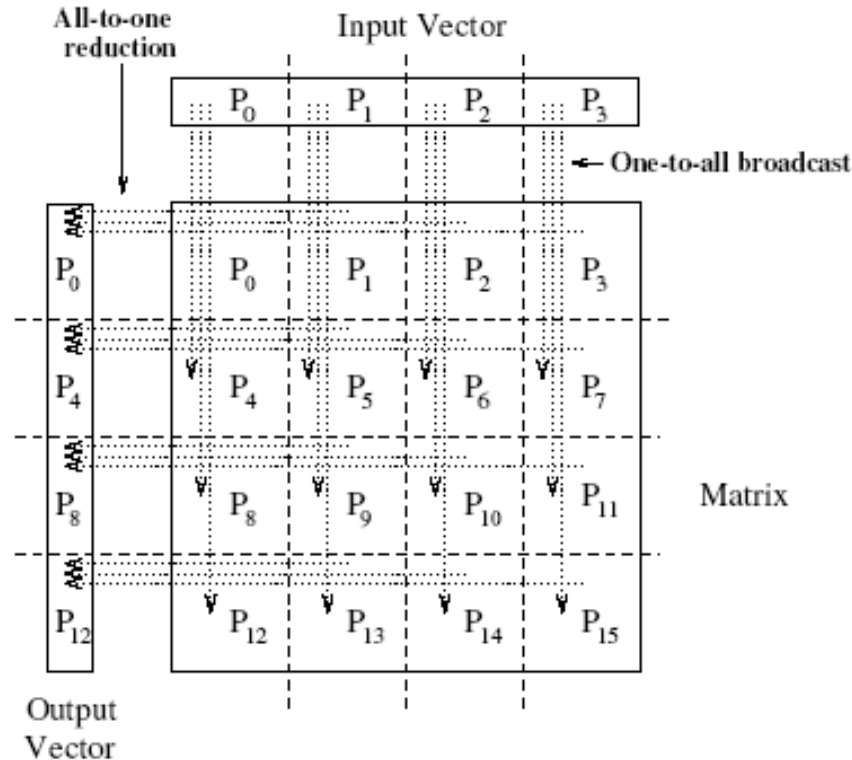
Reduction on an eight-node ring with node 0 as the destination of the reduction.

# Broadcast and Reduction: Example

Consider the problem of multiplying a matrix with a vector.

- The  $n \times n$  matrix is assigned to an  $n \times n$  (virtual) processor grid. The vector is assumed to be on the first row of processors.
- The first step of the product requires a one-to-all broadcast of the vector element along the corresponding column of processors. This can be done concurrently for all  $n$  columns.
- The processors compute local product of the vector element and the local matrix entry.
- In the final step, the results of these products are accumulated to the first processor of each row using  $n$  concurrent all-to-one reduction operations along the rows (using the sum operation).

# Broadcast and Reduction: Matrix-Vector Multiplication Example

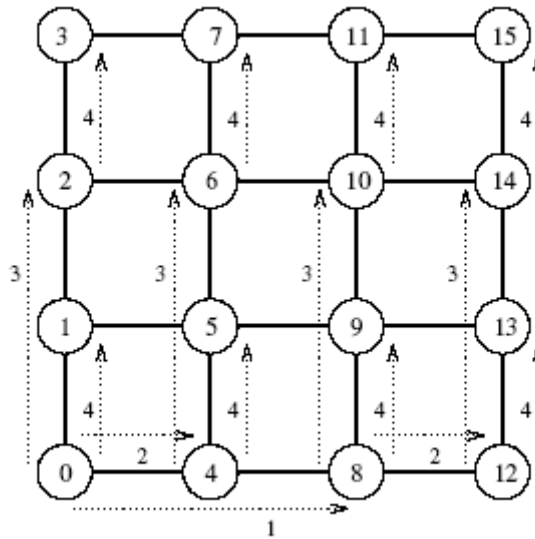


One-to-all broadcast and all-to-one reduction in the multiplication of a  $4 \times 4$  matrix with a  $4 \times 1$  vector.

# Broadcast and Reduction on a Mesh

- We can view each row and column of a square mesh of  $p$  nodes as a linear array of  $\sqrt{p}$  nodes.
- Broadcast and reduction operations can be performed in two steps - the first step does the operation along a row and the second step along each column concurrently.
- This process generalizes to higher dimensions as well.

# Broadcast and Reduction on a Mesh: Example

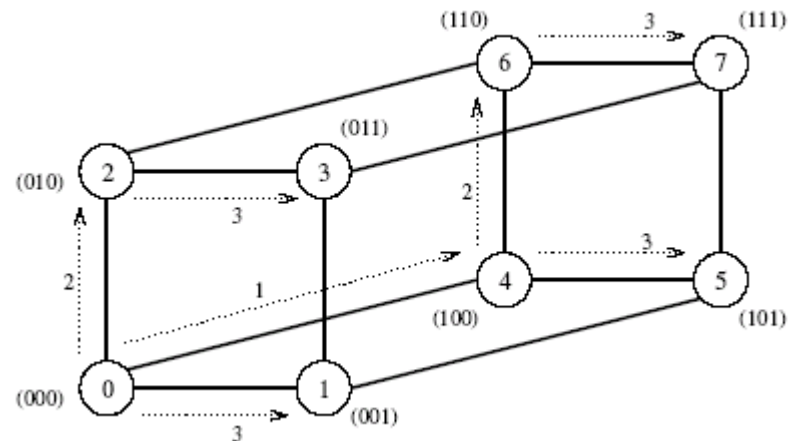


One-to-all broadcast on a 16-node mesh.

# Broadcast and Reduction on a Hypercube

- A hypercube with  $2^d$  nodes can be regarded as a  $d$ -dimensional mesh with two nodes in each dimension.
- The mesh algorithm can be generalized to a hypercube and the operation is carried out in  $d (= \log p)$  steps.

# Broadcast and Reduction on a Hypercube: Example



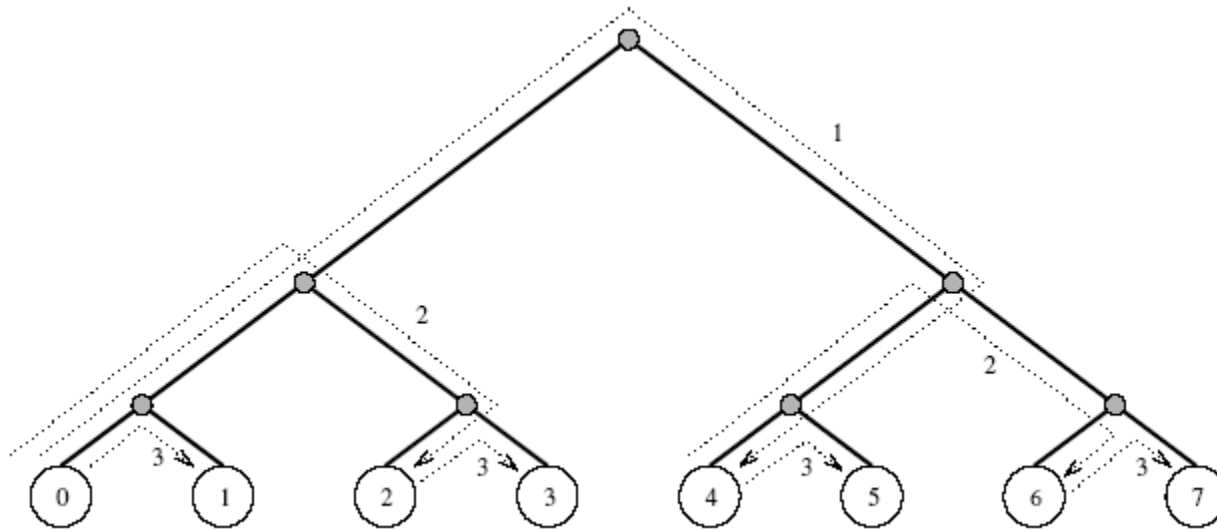
One-to-all broadcast on a three-dimensional hypercube.  
The binary representations of node labels are shown in parentheses.



# Broadcast and Reduction on a Balanced Binary Tree

- Consider a binary tree in which processors are (logically) at the leaves and internal nodes are routing nodes.
- Assume that source processor is the leftmost leaf of this tree. In the first step, the source sends the data to the leftmost leaf of the right subtree of the root. The problem has now been decomposed into two problems with half the number of processors.

# Broadcast and Reduction on a Balanced Binary Tree



One-to-all broadcast on an eight-node tree.

# Broadcast and Reduction Algorithms

- We illustrate the algorithm for a hypercube, but the algorithm can be adapted to other architectures.
- The hypercube has  $2^d$  nodes and *my\_id* is the label for a node.
- *X* is the message to be broadcast, which initially resides at the source node 0.

# Broadcast and Reduction Algorithms

```
1.  procedure GENERAL_ONE_TO_ALL_BC( $d, my\_id, source, X$ )
2.  begin
3.       $my\_virtual\_id := my\_id \text{ XOR } source;$ 
4.       $mask := 2^d - 1;$ 
5.      for  $i := d - 1$  downto 0 do    /* Outer loop */
6.           $mask := mask \text{ XOR } 2^i;$   /* Set bit  $i$  of  $mask$  to 0 */
7.          if ( $my\_virtual\_id \text{ AND } mask$ ) = 0 then
8.              if ( $my\_virtual\_id \text{ AND } 2^i$ ) = 0 then
9.                   $virtual\_dest := my\_virtual\_id \text{ XOR } 2^i;$ 
10.                 send  $X$  to ( $virtual\_dest \text{ XOR } source$ );
11.                /* Convert  $virtual\_dest$  to the label of the physical destination */
12.            else
13.                 $virtual\_source := my\_virtual\_id \text{ XOR } 2^i;$ 
14.                receive  $X$  from ( $virtual\_source \text{ XOR } source$ );
15.                /* Convert  $virtual\_source$  to the label of the physical source */
16.            endelse;
17.        endfor;
18.    end GENERAL_ONE_TO_ALL_BC
```

One-to-all broadcast of a message  $X$  from  $source$  on a hypercube. 20

# Broadcast and Reduction Algorithms

```
1.  procedure ALL_TO_ONE_REDUCE( $d, my\_id, m, X, sum$ )
2.  begin
3.      for  $j := 0$  to  $m - 1$  do  $sum[j] := X[j];$ 
4.       $mask := 0;$ 
5.      for  $i := 0$  to  $d - 1$  do
6.          /* Select nodes whose lower  $i$  bits are 0 */
7.          if  $(my\_id \text{ AND } mask) = 0$  then
8.              if  $(my\_id \text{ AND } 2^i) \neq 0$  then
9.                   $msg\_destination := my\_id \text{ XOR } 2^i;$ 
10.                 send  $sum$  to  $msg\_destination;$ 
11.             else
12.                  $msg\_source := my\_id \text{ XOR } 2^i;$ 
13.                 receive  $X$  from  $msg\_source;$ 
14.                 for  $j := 0$  to  $m - 1$  do
15.                      $sum[j] := sum[j] + X[j];$ 
16.                 endelse;
17.                  $mask := mask \text{ XOR } 2^i;$  /* Set bit  $i$  of  $mask$  to 1 */
18.             endfor;
19.      end ALL_TO_ONE_REDUCE
```

Single-node accumulation on a  $d$ -dimensional hypercube. Each node contributes a message  $X$  containing  $m$  words, and node 0 is the destination.

# Cost Analysis

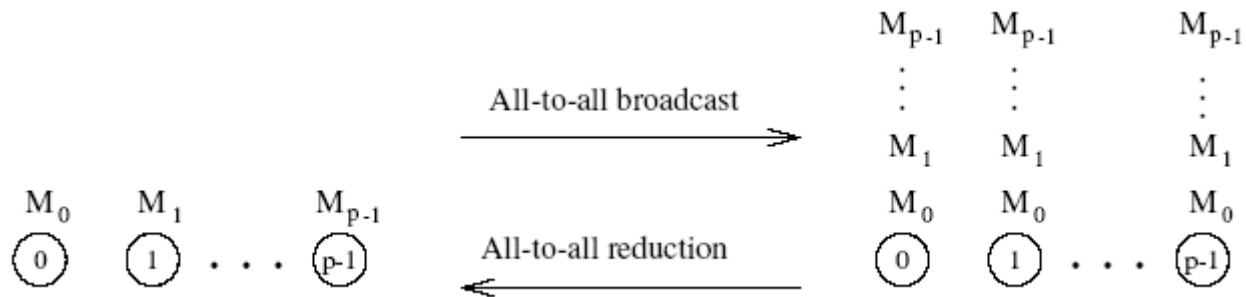
- The broadcast or reduction procedure involves  $\log p$  rounds of point-to-point simple message transfers, each at a time cost of  $t_s + t_w m$ .
- The total time is therefore given by:

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

# All-to-All Broadcast and Reduction

- Generalization of broadcast in which each processor is the source as well as destination.
- A process sends the same  $m$ -word message to every other process, but different processes may broadcast different messages.

# All-to-All Broadcast and Reduction



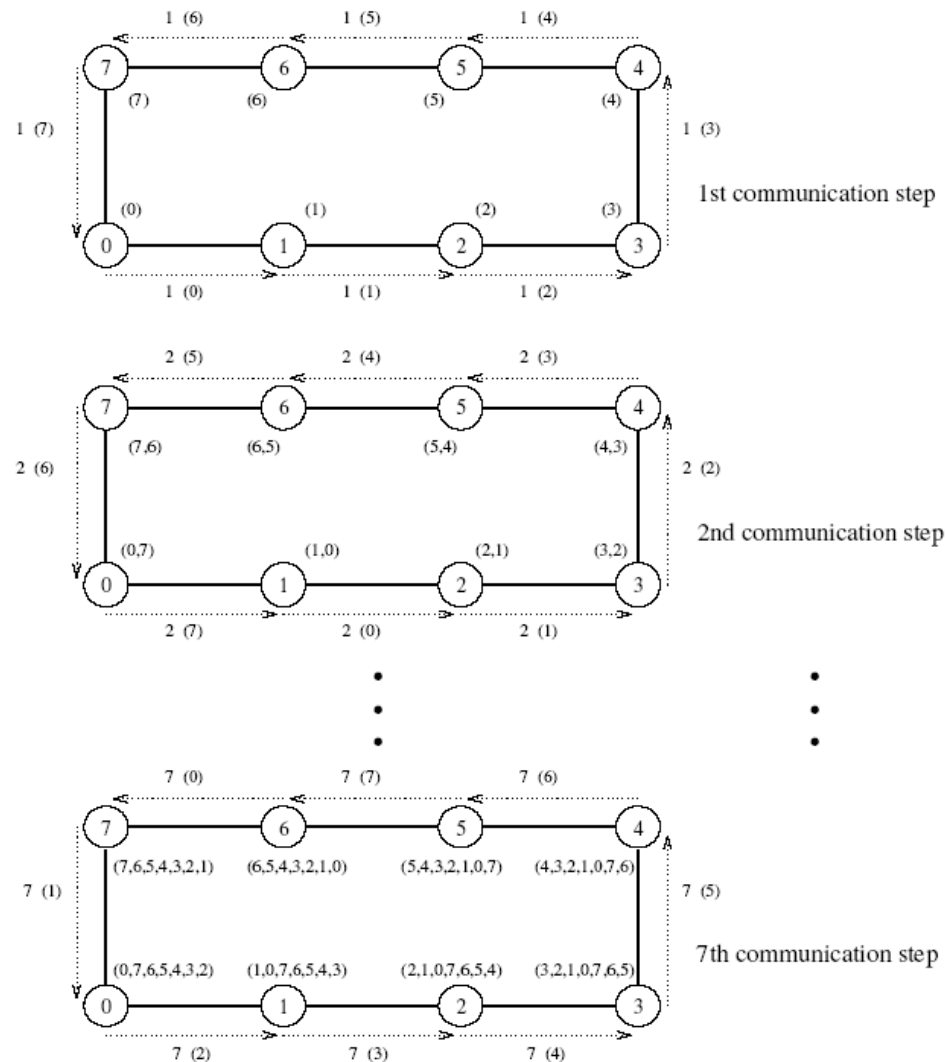
All-to-all broadcast and all-to-all reduction.



# All-to-All Broadcast and Reduction on a Ring

- Simplest approach: perform  $p$  one-to-all broadcasts. This is not the most efficient way, though. Better as follows:
- 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.
- The algorithm terminates in  $p-1$  steps.

# All-to-All Broadcast and Reduction on a Ring



All-to-all broadcast on an eight-node ring.

# All-to-All Broadcast on a Ring

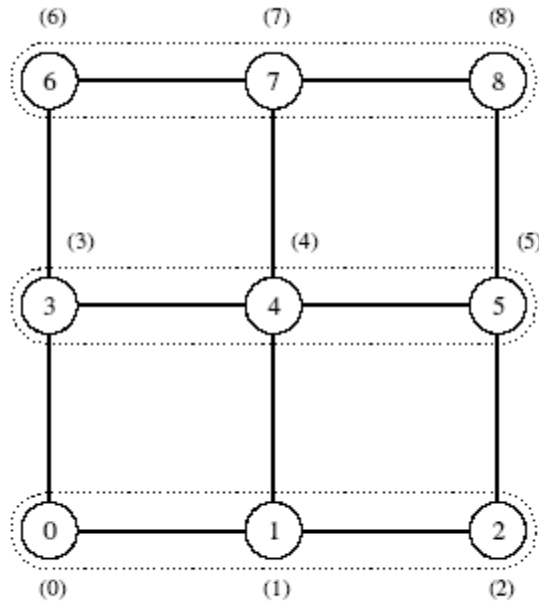
```
1.  procedure ALL_TO_ALL_BC_RING(my_id, my_msg, p, result)
2.  begin
3.      left := (my_id - 1) mod p;
4.      right := (my_id + 1) mod p;
5.      result := my_msg;
6.      msg := result;
7.      for i := 1 to p - 1 do
8.          send msg to right;
9.          receive msg from left;
10.       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

All-to-all broadcast on a  $p$ -node ring.

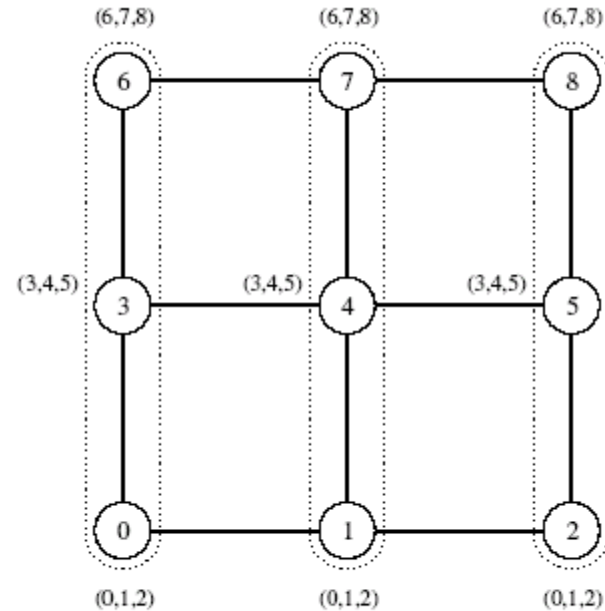
# All-to-all Broadcast on a Mesh

- Performed 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  $m\sqrt{p}$ .
- The second communication phase is a columnwise all-to-all broadcast of the consolidated messages.

# All-to-all Broadcast on a Mesh



(a) Initial data distribution



(b) Data distribution after rowwise broadcast

All-to-all broadcast on a 3 x 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).

# All-to-all Broadcast on a Mesh

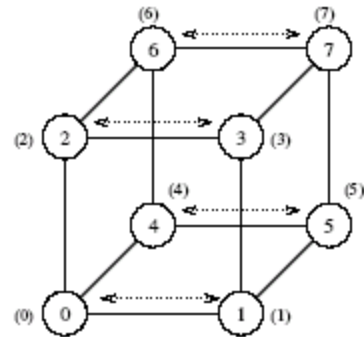
```
1.      procedure ALL_TO_ALL_BC_MESH(my_id, my_msg, p, result)
2.      begin
3.      /* Communication along rows */
4.      left := my_id - (my_id mod  $\sqrt{p}$ ) + (my_id - 1) mod  $\sqrt{p}$ ;
5.      right := my_id - (my_id mod  $\sqrt{p}$ ) + (my_id + 1) mod  $\sqrt{p}$ ;
6.      result := my_msg;
7.      msg := result;
8.      for i := 1 to  $\sqrt{p} - 1$  do
9.          send msg to right;
10.         receive msg from left;
11.         result := result  $\cup$  msg;
12.     endfor;
13. /* Communication along columns */
14. up := (my_id -  $\sqrt{p}$ ) mod p;
15. down := (my_id +  $\sqrt{p}$ ) mod p;
16. msg := result;
17. for i := 1 to  $\sqrt{p} - 1$  do
18.     send msg to down;
19.     receive msg from up;
20.     result := result  $\cup$  msg;
21. endfor;
22. end ALL_TO_ALL_BC_MESH
```

All-to-all broadcast on a square mesh of  $p$  nodes.

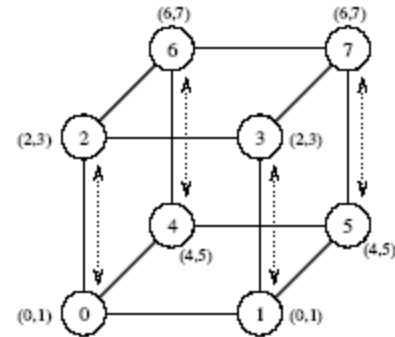
# All-to-all broadcast on a Hypercube

- Generalization of the mesh algorithm to  $\log p$  dimensions.
- Message size doubles at each of the  $\log p$  steps.

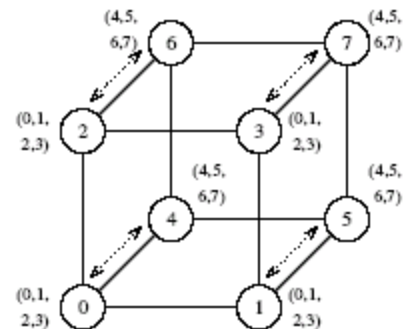
# All-to-all broadcast on a Hypercube



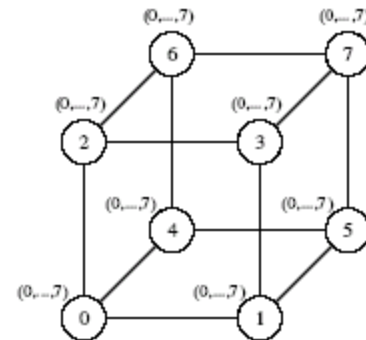
(a) Initial distribution of messages



(b) Distribution before the second step



(c) Distribution before the third step



(d) Final distribution of messages

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



# All-to-all broadcast on a 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  $\cup$  msg;
9.      endfor;
10. end ALL_TO_ALL_BC_HCUBE
```

All-to-all broadcast on a  $d$ -dimensional hypercube.

# All-to-all Reduction

- Similar communication pattern to all-to-all broadcast, except in the reverse order.
- On 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.

# Cost Analysis

- On a ring, the time is given by:  $(t_s + t_w m)(p-1)$ .
- On a mesh, the time is given by:  $2t_s(\sqrt{p} - 1) + t_w m(p-1)$ .
  - Phase 1:  $(t_s + t_w m)(\sqrt{p}-1)$
  - Phase 2:  $(t_s + t_w m\sqrt{p})(\sqrt{p}-1)$  //each message sizes  $m\sqrt{p}$
- On a hypercube, we have:

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

These times are asymptotically optimal in message size.

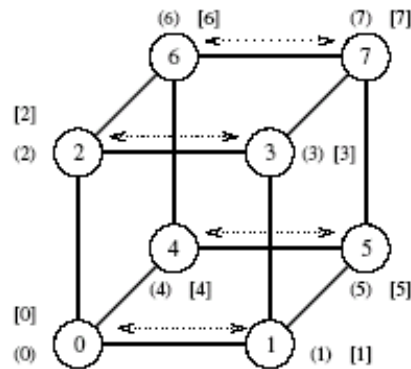
# The All-Reduce Operation

- In all-reduce, 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.
- All-reduce is identical to all-to-one reduction followed by a one-to-all broadcast. This formulation is not the most efficient.
- A more efficient way is to use the pattern of all-to-all broadcast. The only difference is to sum up the numbers (reduce) instead of accumulating messages. Time for this All-Reduce is  $(t_s + t_w m) \log p$ .

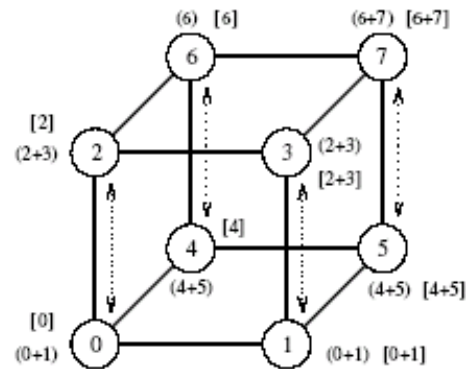
# The Prefix-Sum Operation

- Given  $p$  numbers  $n_0, n_1, \dots, n_{p-1}$  (one on each node), the problem is to compute the sums  $s_k = \sum_{i=0}^k n_i$  for all  $k$  between 0 and  $p-1$ .
- Initially,  $n_k$  resides on the node labeled  $k$ , and at the end of the procedure, the same node holds  $S_k$ .

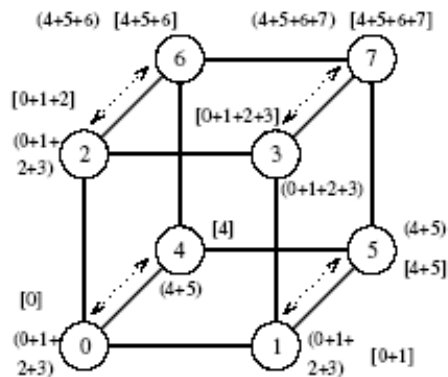
# The Prefix-Sum Operation



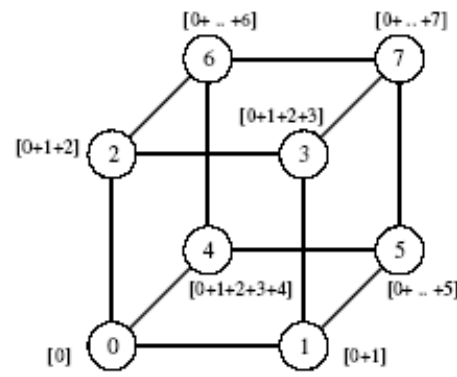
(a) Initial distribution of values



(b) Distribution of sums before second step



(c) Distribution of sums before third step



(d) Final distribution of prefix sums

Computing prefix sums on an eight-node hypercube.

Square brackets show the local prefix sum accumulated in the result buffer.

Parentheses enclose the outgoing message buffer for the next step.

# The Prefix-Sum Operation

- The operation can be implemented using the all-to-all broadcast kernel.
- We must account for the fact that in prefix sums the node with label  $k$  uses information from only the  $k$ -node subset whose labels are less than or equal to  $k$ .
- This is implemented using an additional result buffer. 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 the recipient node.
- The contents of the outgoing message (denoted by parentheses in the figure) are updated with every incoming message.

# The Prefix-Sum Operation

```
1.  procedure PREFIX_SUMS_HCUBE(my_id, my_number, d, result)
2.  begin
3.      result := my_number;
4.      msg := result;
5.      for i := 0 to d - 1 do
6.          partner := my_id XOR  $2^i$ ;
7.          send msg to partner;
8.          receive number from partner;
9.          msg := msg + number;
10.         if (partner < my_id) then result := result + number;
11.     endfor;
12. end PREFIX_SUMS_HCUBE
```

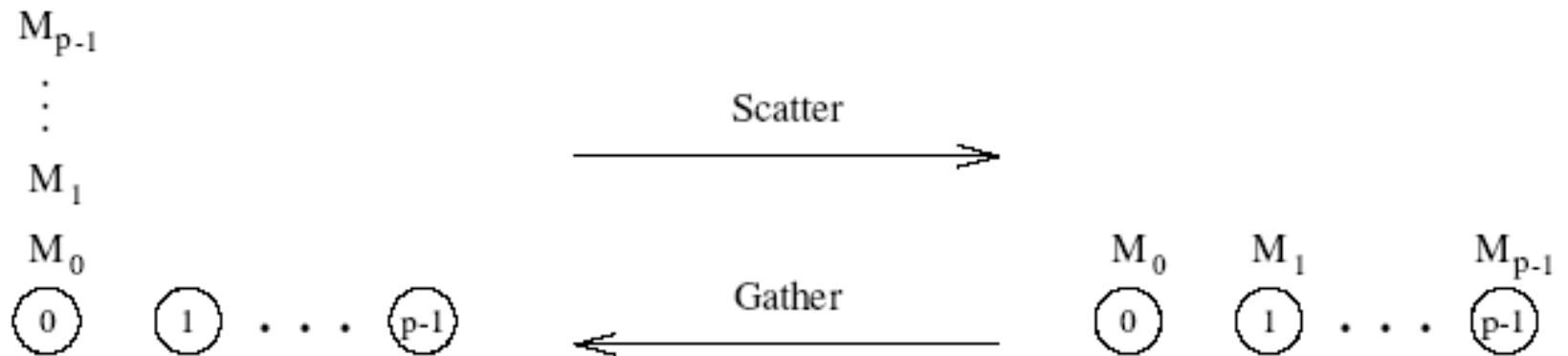
Prefix sums on a  $d$ -dimensional hypercube.



# Scatter and Gather

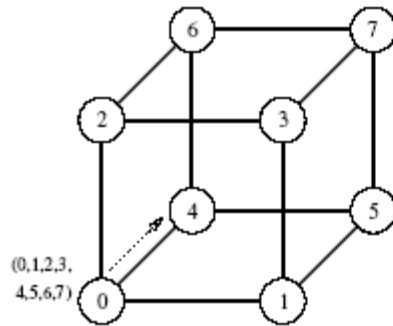
- In the *scatter* operation, a single node sends a unique message of size  $m$  to every other node (also called a one-to-all personalized communication). In other words, each node starts with  $p$  unique messages, each destined for one of the  $p$  nodes.
- In the *gather* operation, a single node collects a unique message from each node.
- While the scatter operation is fundamentally different from broadcast, the algorithmic structure is similar, except for differences in message sizes (messages get smaller in scatter and stay constant in broadcast).
- The gather operation is exactly the inverse of the scatter operation and can be executed as such.

# Gather and Scatter Operations

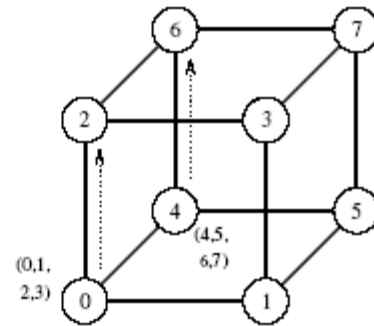


Scatter and gather operations.

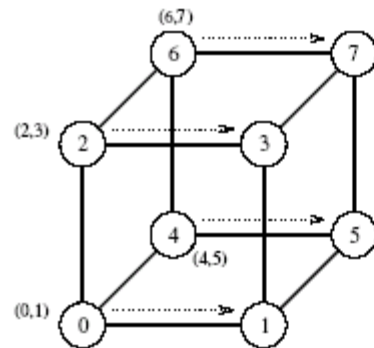
# Example of the Scatter Operation



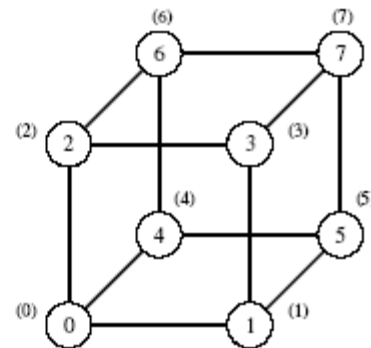
(a) Initial distribution of messages



(b) Distribution before the second step



(c) Distribution before the third step



(d) Final distribution of messages

The scatter operation on an eight-node hypercube.

# Cost of Scatter and Gather

- There are  $\log p$  steps, in each step, the number of nodes halves and the data size halves.
- We have the time for this operation to be:

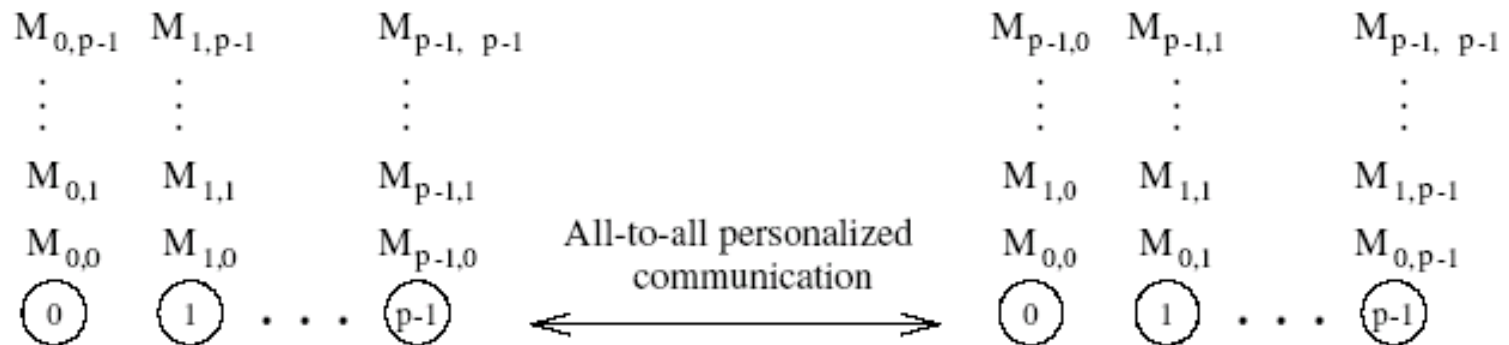
$$T = t_s \log p + t_w m(p - 1).$$

- This time holds for a linear array as well as a 2-D mesh.
- These times are asymptotically optimal in message size.

# All-to-All Personalized Communication

- Each node has a distinct message of size  $m$  for every other node.
- This is unlike all-to-all broadcast, in which each node sends the same message to all other nodes.
- All-to-all personalized communication is also known as *total exchange*.

# All-to-All Personalized Communication

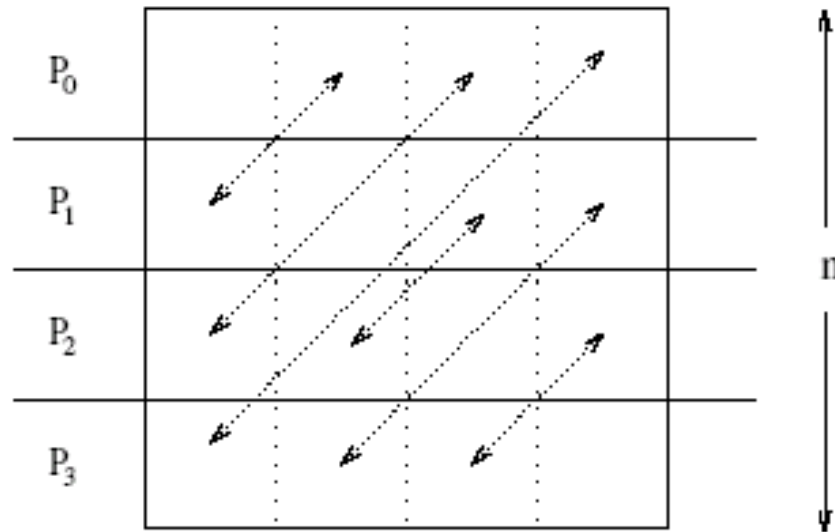


All-to-all personalized communication.

# All-to-All Personalized Communication: Example

- Consider the problem of transposing a matrix.
- Each processor contains one full row of the matrix.
- The transpose operation in this case is identical to an all-to-all personalized communication operation.

# All-to-All Personalized Communication: Example



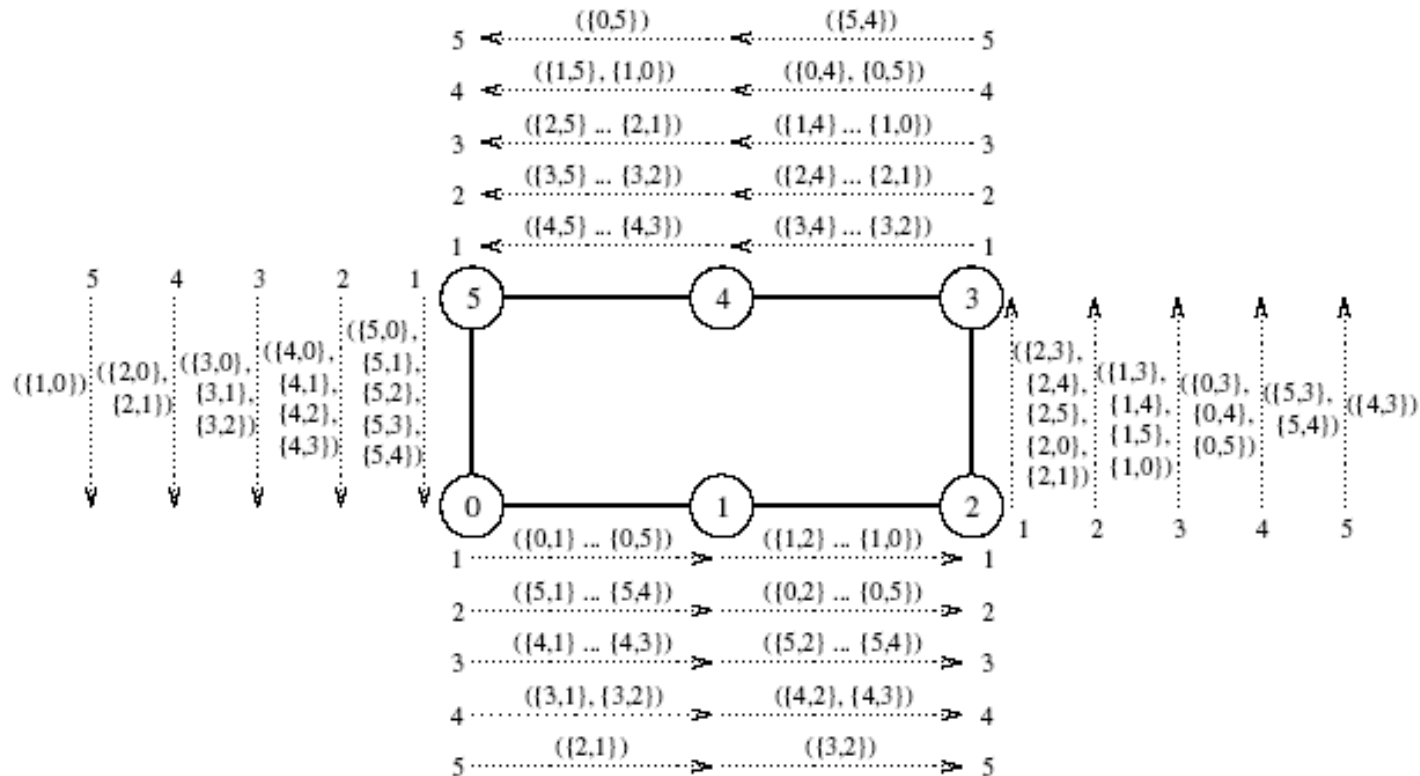
All-to-all personalized communication in transposing a  $4 \times 4$  matrix using four processes.



# All-to-All Personalized Communication on a Ring

- Each node sends all pieces of data as one consolidated message of size  $m(p - 1)$  to one of its neighbors.
- 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.
- The algorithm terminates in  $p - 1$  steps.
- The size of the message reduces by  $m$  at each step.

# All-to-All Personalized Communication on a Ring



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  $\{\{x_1, y_1\}, \{x_2, y_2\}, \dots, \{x_n, y_n\}\}$  indicates a message that is formed by concatenating  $n$  individual messages.

# All-to-All Personalized Communication on a Ring: Cost

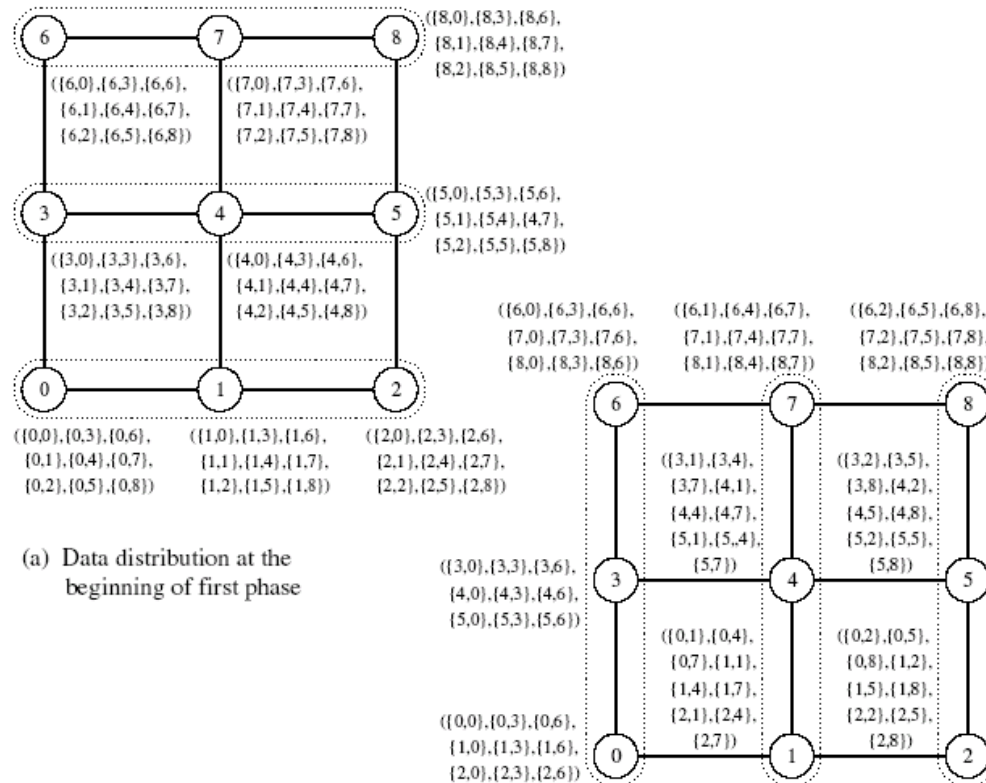
- We have  $p - 1$  steps in all.
- In step  $i$ , the message size is  $m(p - i)$ .
- The total time is given by:

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

# All-to-All Personalized Communication on a Mesh

- Each node first groups its  $p$  messages according to the columns of their destination nodes.
- All-to-all personalized communication is performed independently in each row with clustered messages of size  $m\sqrt{p}$ .
- Messages in each node are sorted again, this time according to the rows of their destination nodes.
- All-to-all personalized communication is performed independently in each column with clustered messages of size  $m\sqrt{p}$ .

# All-to-All Personalized Communication on a Mesh



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

# All-to-All Personalized Communication on a Mesh: Cost

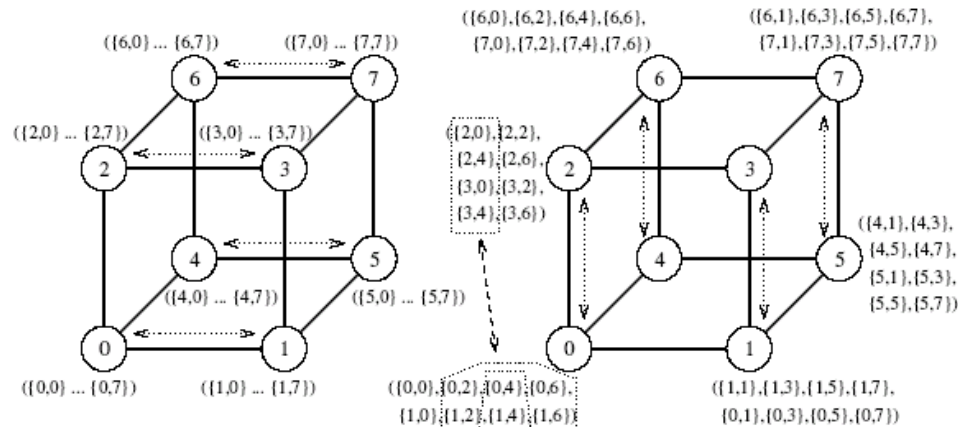
- Time for the first phase is identical to that in a ring with  $\sqrt{p}$  processors, i.e.,  $(t_s + t_w mp/2)(\sqrt{p} - 1)$ .
- Time in the second phase is identical to the first phase. Therefore, total time is twice of this time, i.e.,

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

# All-to-All Personalized Communication on a Hypercube

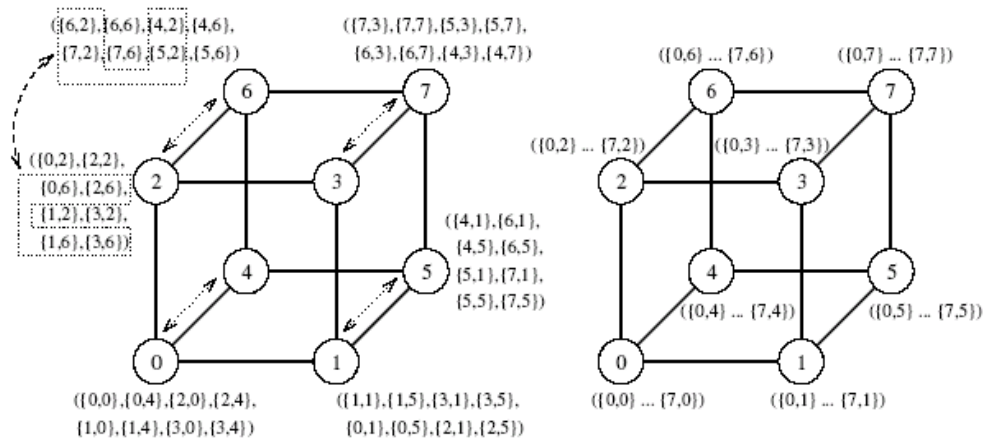
- Generalize the mesh algorithm to  $\log p$  steps.
- At any stage in all-to-all personalized communication, every node holds  $p$  packets of size  $m$  each.
- While communicating in a particular dimension, every node sends  $p/2$  of these packets (consolidated as one message).
- A node must rearrange its messages locally before each of the  $\log p$  communication steps.

# All-to-All Personalized Communication on a Hypercube



(a) Initial distribution of messages

(b) Distribution before the second step



(c) Distribution before the third step

(d) Final distribution of messages

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



# All-to-All Personalized Communication on a Hypercube: Cost

- We have  $\log p$  iterations and  $mp/2$  words are communicated in each iteration. Therefore, the cost is:

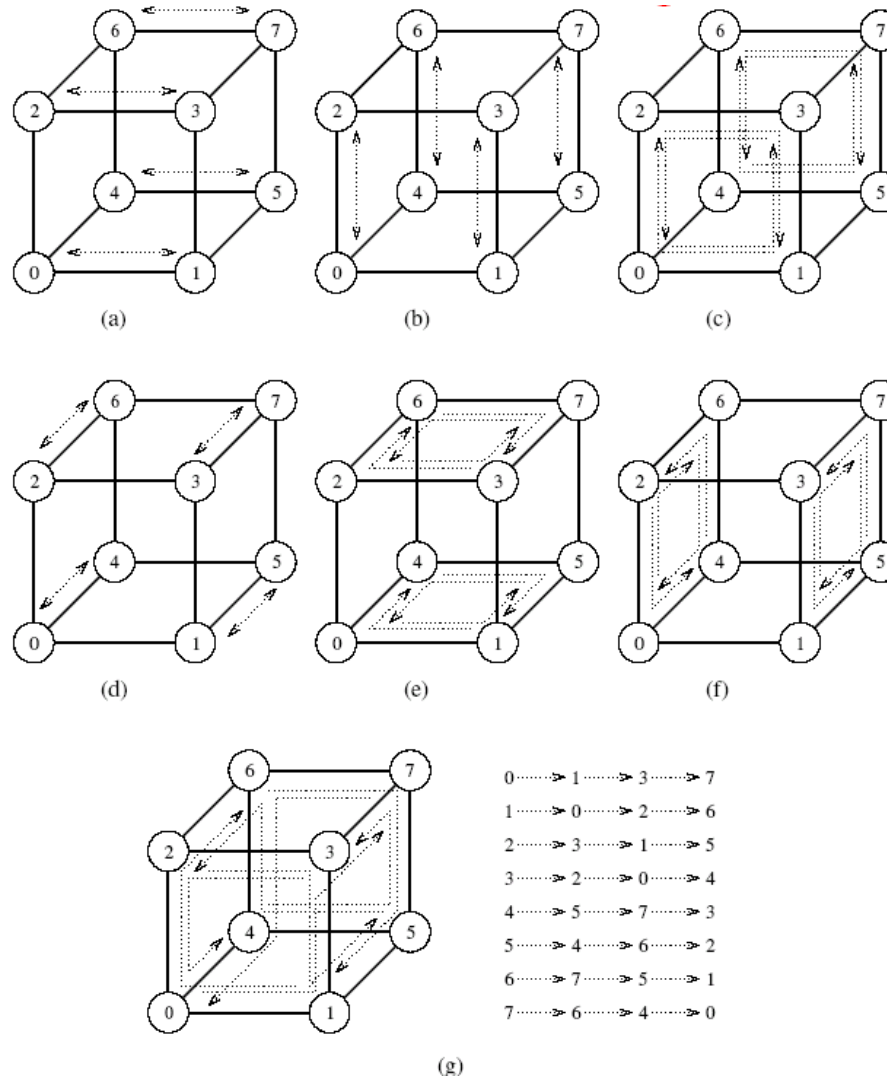
$$T = (t_s + t_w mp/2) \log p.$$

- This is not optimal!

# All-to-All Personalized Communication on a Hypercube: Optimal Algorithm

- 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.
- In the  $j^{th}$  communication step, node  $i$  exchanges data with node  $(i \text{ XOR } j)$ .
- In this schedule, all paths in every communication step are congestion-free, and none of the bidirectional links carry more than one message in the same direction.
- The routing scheme is called E-cube routing.

# All-to-All Personalized Communication on a Hypercube: Optimal Algorithm



# All-to-All Personalized Communication on a Hypercube: Optimal Algorithm

```
1.      procedure ALL_TO_ALL_PERSONAL( $d, my\_id$ )
2.      begin
3.          for  $i := 1$  to  $2^d - 1$  do
4.              begin
5.                   $partner := my\_id \text{ XOR } i$ ;
6.                  send  $M_{my\_id, partner}$  to  $partner$ ;
7.                  receive  $M_{partner, my\_id}$  from  $partner$ ;
8.              endfor;
9.      end ALL_TO_ALL_PERSONAL
```

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  $j$ .

# All-to-All Personalized Communication on a Hypercube: Cost Analysis of Optimal Algorithm

- There are  $p - 1$  steps and each step involves non-congesting message transfer of  $m$  words.
- We have:

$$T_{\text{=}}(t_s + t_w m)(p - 1).$$

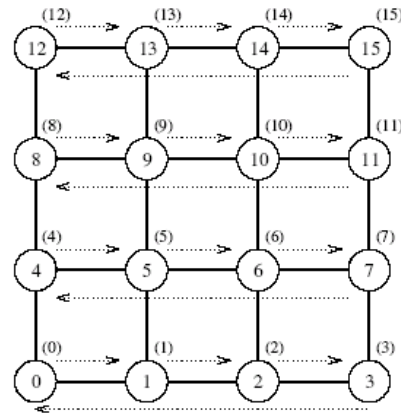
- This is asymptotically optimal in message size.

# Circular Shift

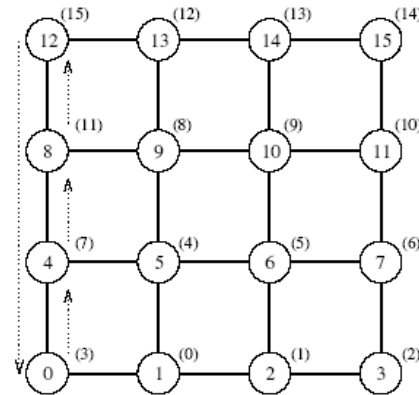
- A special permutation in which node  $i$  sends a data packet to node  $(i + q) \bmod p$  in a  $p$ -node ensemble ( $0 \leq q \leq p$ ).
- The implementation on a ring is rather intuitive. It can be performed in  $\min\{q, p - q\}$  neighbor communications.
- Mesh algorithms follow from this as well. We shift in one direction (all processors) followed by the next direction.
- The associated time has an upper bound of:

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

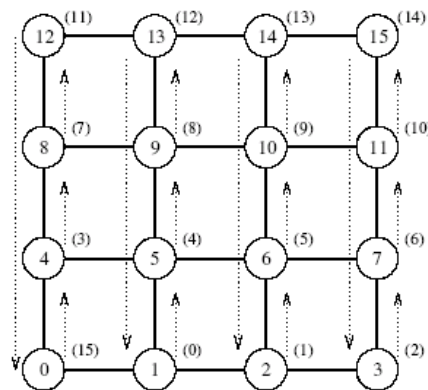
# Circular Shift on a Mesh



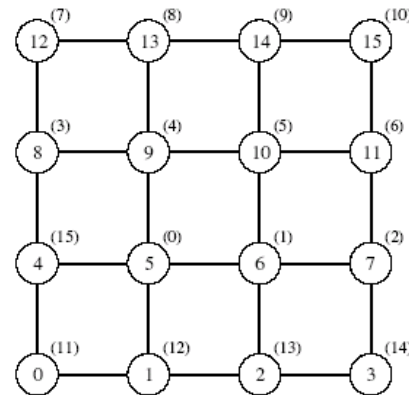
(a) Initial data distribution and the first communication step



(b) Step to compensate for backward row shifts



(c) Column shifts in the third communication step



(d) Final distribution of the data

The communication steps in a circular 5-shift on a 4 x 4 mesh.

# Circular Shift on a Hypercube

- Map a linear array with  $2^d$  nodes onto a  $d$ -dimensional hypercube with node id in reflected Gray code (RGC).
- A property of this mapping is that any two nodes at a distance of  $2^i$  in the array has a distance of  $2^i$  in the cube
- To perform a  $q$ -shift, we expand  $q$  as a sum of  $s$  distinct powers of 2. The circular  $q$ -shift on a hypercube is performed in  $s$  phases with each phase shifting two steps (distance of 2) except the last phase only one step.
- The time is upper bounded by:

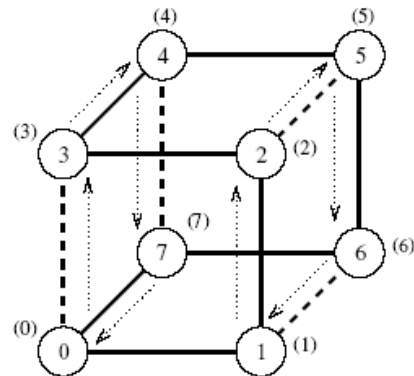
$$T = (t_s + t_w m)(2 \log p - 1).$$

If E-cube routing is used, this time can be reduced to

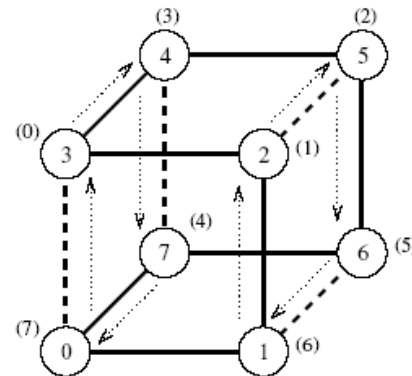
$$T = t_s + t_w m.$$



# Circular Shift on a Hypercube

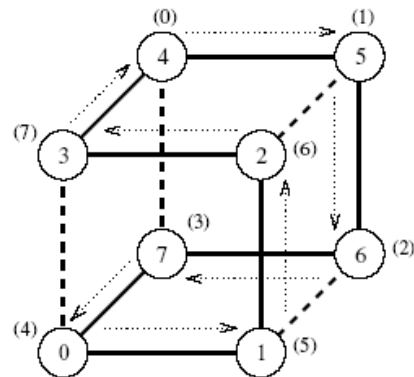


First communication step of the 4-shift

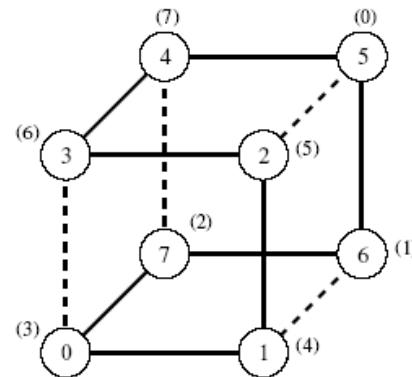


Second communication step of the 4-shift

(a) The first phase (a 4-shift)



(b) The second phase (a 1-shift)



(c) Final data distribution after the 5-shift

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. 65

# Improving Performance of Operations

- Splitting and routing messages into parts: If the message can be split into  $p$  parts, a one-to-all broadcast can be implemented as a scatter operation followed by an all-to-all broadcast operation. The time for this is:

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

- All-to-one reduction can be performed by performing all-to-all reduction (dual of all-to-all broadcast) followed by a gather operation (dual of scatter).

# Improving Performance of Operations

- 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 can be used to construct a similar algorithm for the all-reduce operation.

# Summary

- Assume the algorithm most suitable for the given message size is chosen. These time bounds are valid for any architecture with a  $\Theta(p)$  cross-section bandwidth.

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