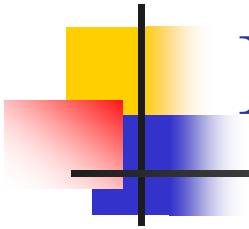


Lecture Outline

Introduction

4.1 One-to-All Broadcast and All-to-One Reduction

4.2 All-to-All Broadcast and Reduction

4.3 All-Reduce and Prefix-Sum Operations

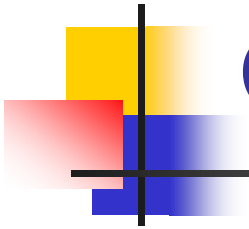
4.4 Scatter and Gather

4.5 All-to-All Personalized Communication

4.6 Circular Shift

4.7 Improving the Speed of Some Communication Operation

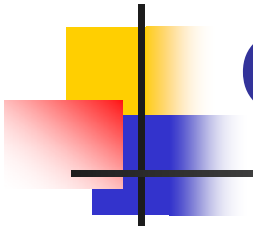
4.8 Summary



Basic Communication Operations: Introduction (1/2)

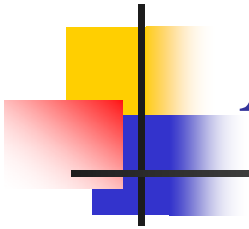
- Many interactions in practical parallel programs occur in well-defined patterns involving groups of processors.
- Efficient implementations of these operations can improve performance, reduce development effort and cost, and improve software quality.
- Efficient implementations must leverage underlying architecture. For this reason, we refer to specific architectures here.
- We select a descriptive set of architectures to illustrate the process of algorithm design.

Basic Communication Operations: Introduction (2/2)



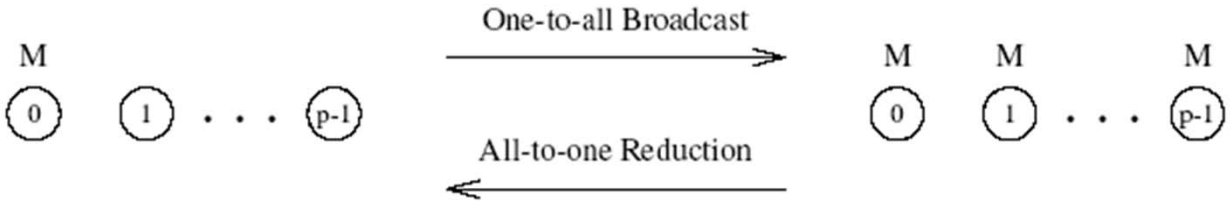
- Group communication operations are built using point-to-point messaging primitives.
- Recall from our discussion of architectures that communicating a message of size m over an non-congested network takes time $t_s + mt_w$.
- We use this as the basis for our analyses. Where necessary, we take congestion into account explicitly by scaling the t_w term.
- We assume that the network is bidirectional and that communication is single-ported.

4.1 One-to-All Broadcast and All-to-One Reduction (1/2)



- 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.

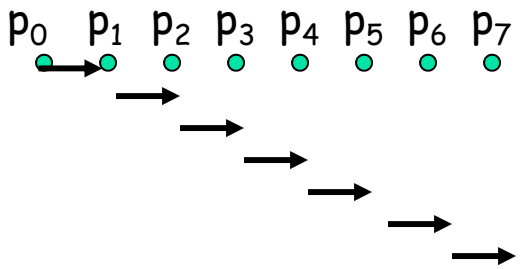
One-to-All Broadcast and All-to-One Reduction (2/2)



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

4.1.1.1 One-to-All Broadcast on Rings (1/3)

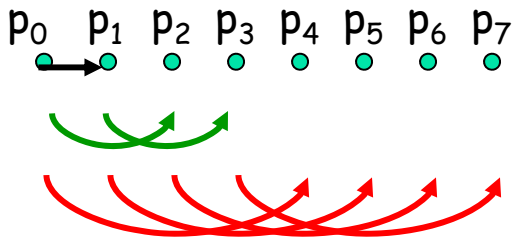
- Naïve Approach
  - p-1 messages sent sequentially from the source to the other *p-1* processors
  - not very efficient.
  - terrible complexity $(t_s+mt_w)(p-1)$

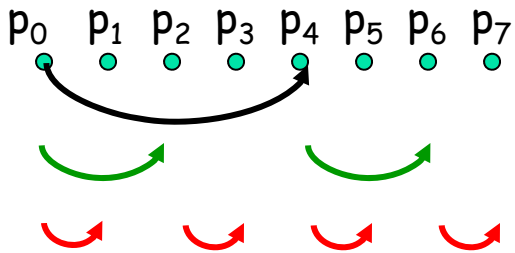


Ref CommImpl1.ppt

One-to-All Broadcast on Rings (2/3)

- Recursive doubling
  - use logical binary tree for broadcasting
  - make sure to minimize congestion
  - complexity: $(t_s+t_{wm})logp$



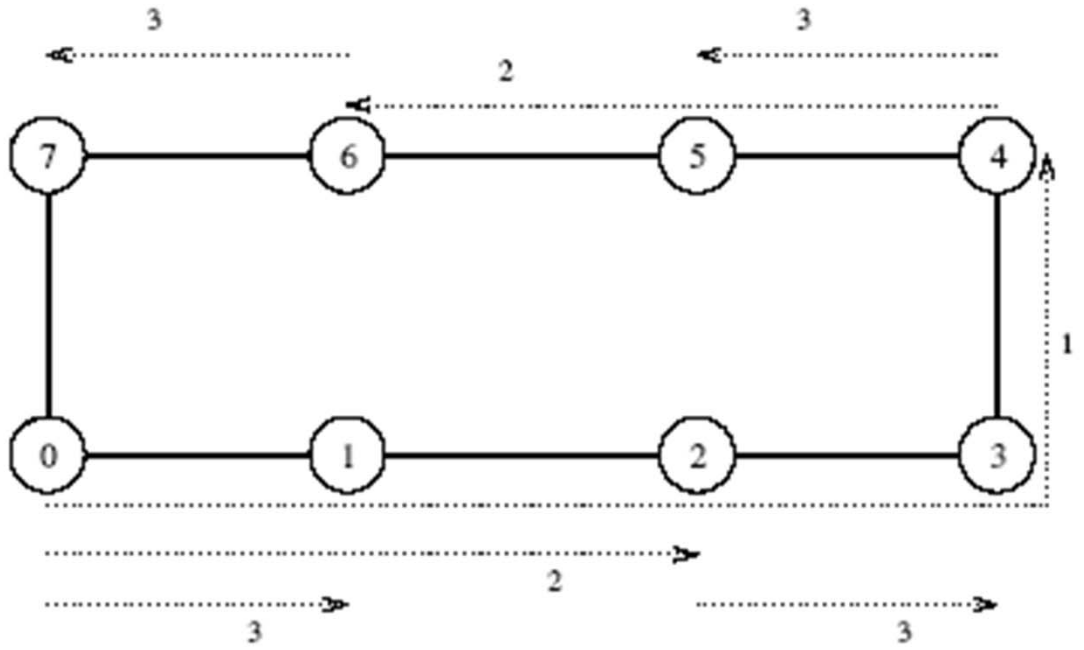


Which better?

Ref CommImpl1.ppt

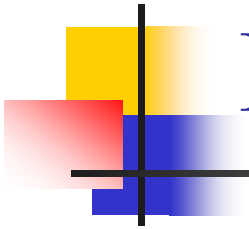
One-to-All Broadcast on Rings

(3/3)



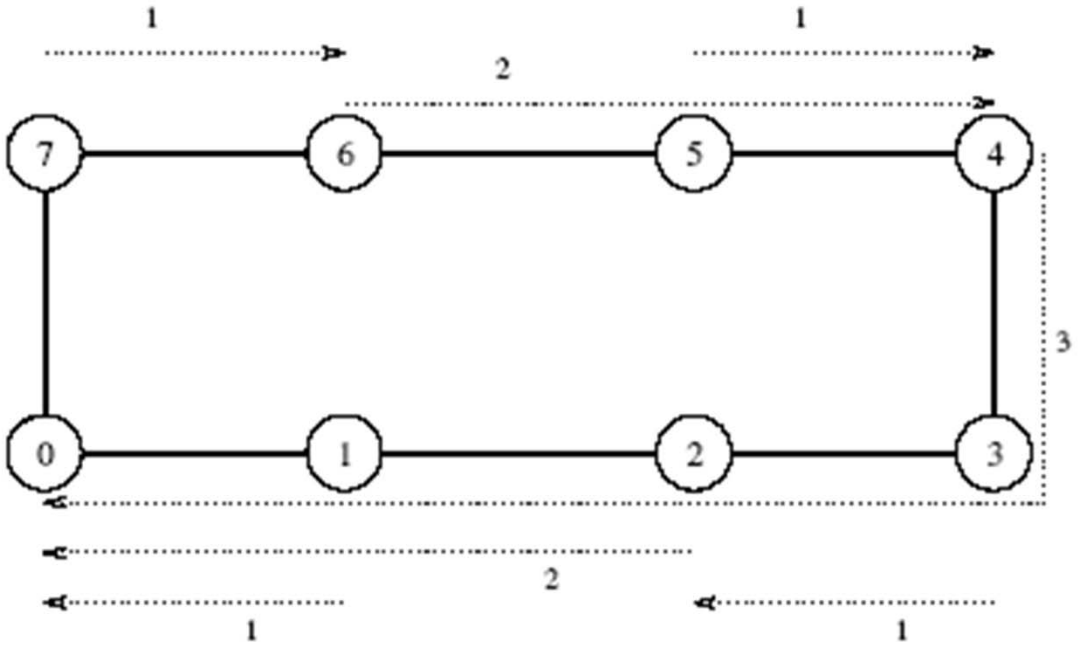
One-to-all broadcast on an eight-node ring. Node 0 is the source of the broadcast. Each message transfer step is shown by a numbered, dotted arrow from the source of the message to its destination. The number on an arrow indicates the time step during which the message is transferred.

4.1.1.2 All-to-One Reduction on Rings (1/2)



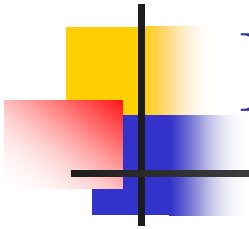
- Reduction can be performed in an identical fashion by inverting the process.

All-to-One Reduction on Rings (2/2)



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

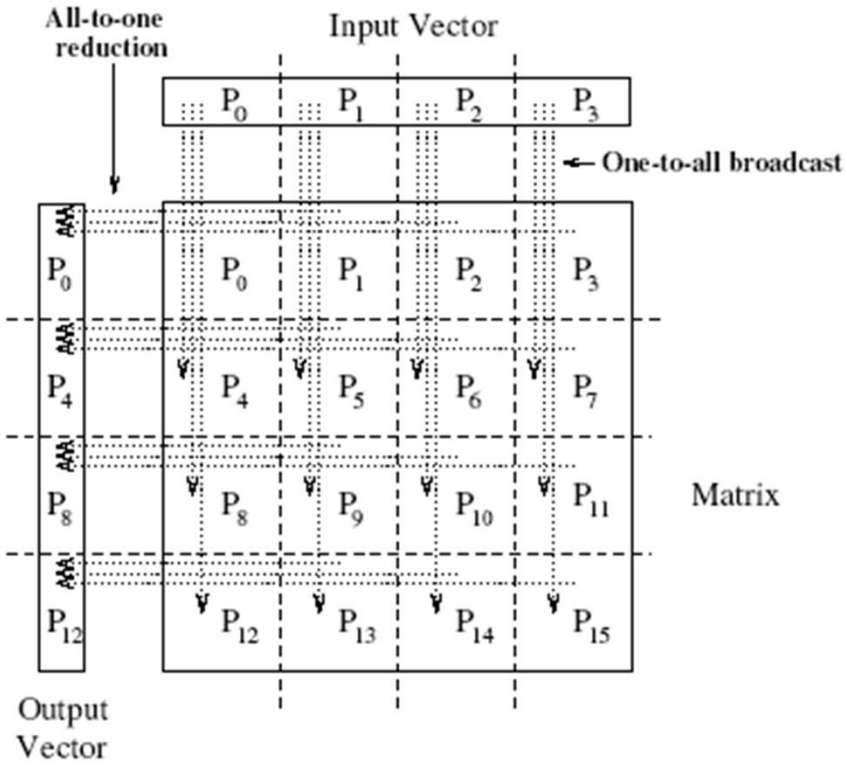
Broadcast and Reduction on Rings : Example (1/2)



Consider the problem of multiplying a matrix with a vector.

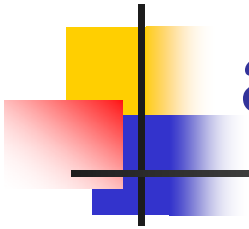
- The n x n matrix is assigned to an n x n (virtual) processor grid. The vector is assumed to be on the first row of processors.
- The first step of the product requires a one-to-all broadcast of the vector element along the corresponding column of processors. This can be done concurrently for all n 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 row using n concurrent all-to-one reduction operations along the columns (using the sum operation).

Broadcast and Reduction on Rings Example : Matrix-Vector Multiplication (2/2)



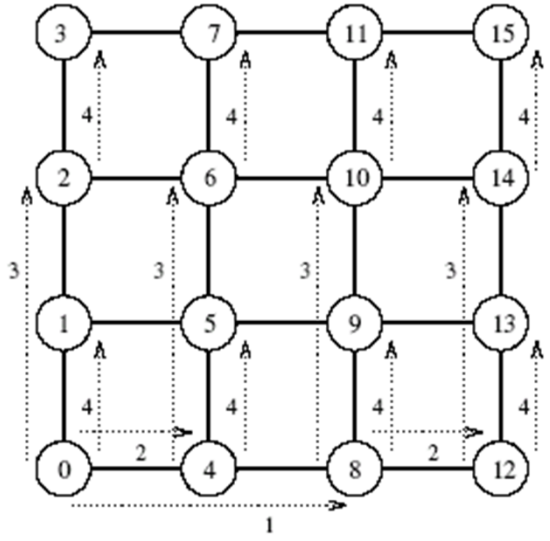
One-to-all broadcast and all-to-one reduction in the multiplication of a 4 x 4 matrix with a 4 x 1 vector.

4.1.2 Broadcast and Reduction on a Mesh (1/2)



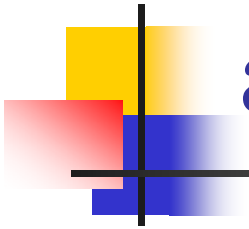
- Each row and column of mesh is a ring! $\sqrt{p}$ nodes
- Two phases
  - Ring broadcast in row of initiating node
  - Ring broadcast in all columns from row of initiating node
- Similar *(n phase)* procedure works on *nD* mesh
- Matrix Vector product
  - *n*n* matrix, *n*1* vector, *n*n* mesh

Broadcast and Reduction on a Mesh: Example (2/2)



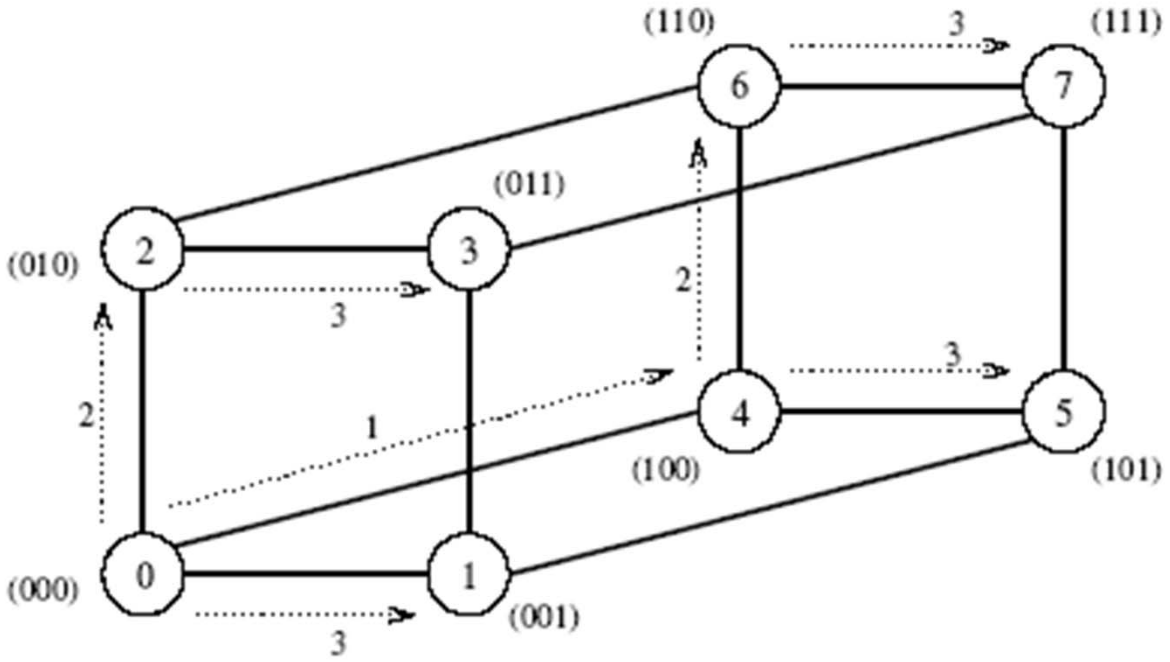
One-to-all broadcast on a 16-node mesh.

4.1.3 Broadcast and Reduction on a Hypercube (1/2)



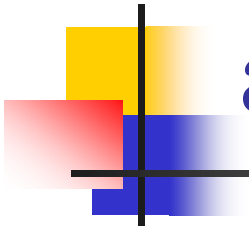
- *d* phases on *d*-dimension hypercube
- phase i:
  - $2^{(i-1)}$ messages
  - sent in parallel
  - in dimension i
- source node = 000, d = 3
  - Phase 1: 000 -> 100
  - Phase 2: 000 -> 010 100 -> 110
  - Phase 3: 000 -> 001 100 -> 101
    010 -> 011 110 -> 111

Broadcast and Reduction on a Hypercube: Example (2/2)



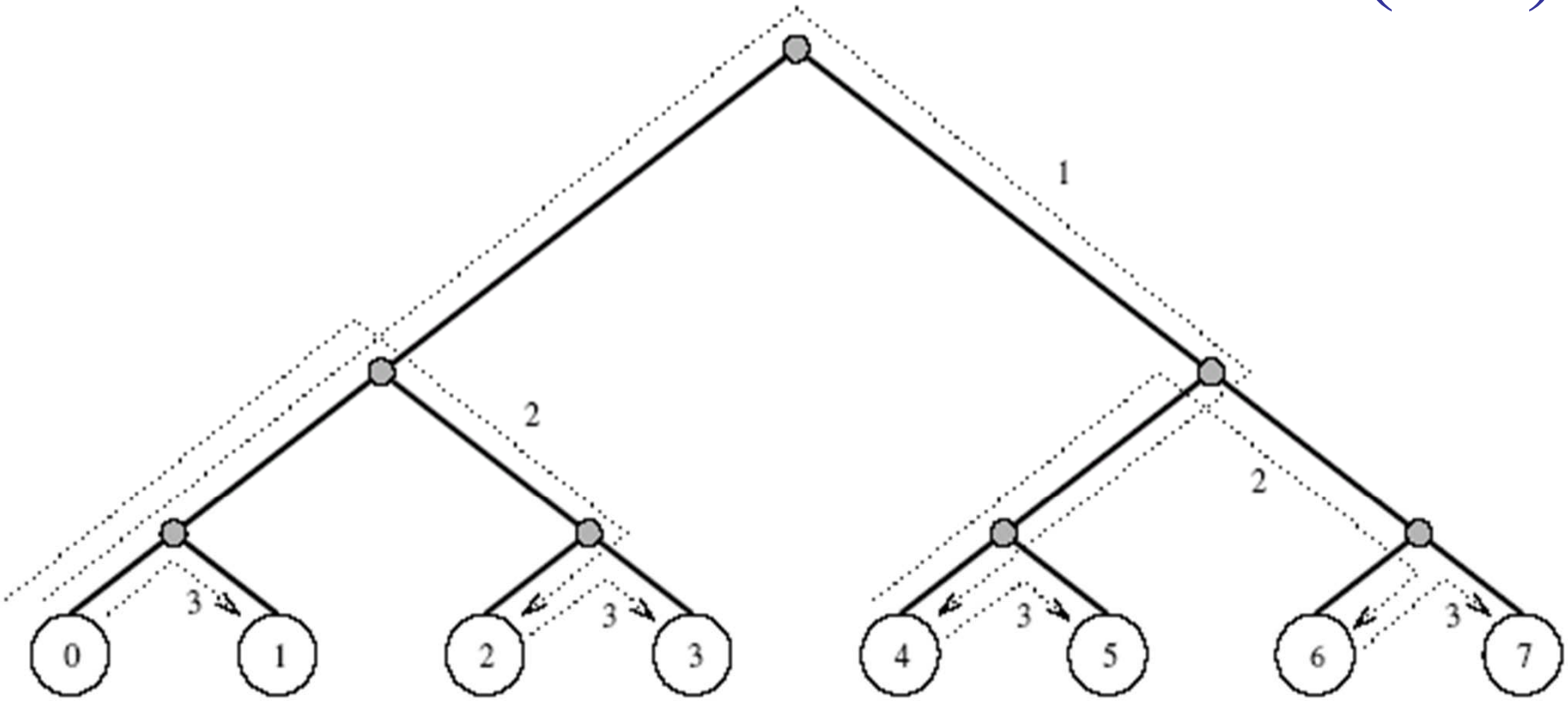
One-to-all broadcast on a three-dimensional hypercube. The binary representations of node labels are shown in parentheses.

4.1.4 Broadcast and Reduction on a Balanced Binary Tree (1/2)



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

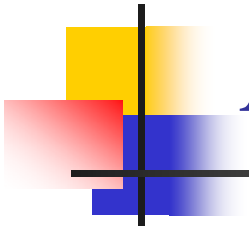
Broadcast and Reduction on a Balanced Binary Tree (2/2)



Source

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

4.1.4 Broadcast and Reduction Algorithms (1/3)



- All of the algorithms described above are adaptations of the same algorithmic template.
- We illustrate the algorithm for a hypercube, but the algorithm, as has been seen, 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 Algorithm

(2/3)

```
1.   procedure GENERAL_ONE_TO_ALL_BC(d, my_id, source, X)
2.   begin
3.       my_virtual_id := my_id XOR source;
4.       mask := 2^d − 1;
5.       for i := d − 1 downto 0 do      /* Outer loop */
6.           mask := mask XOR 2^i;    /* Set bit i of mask to 0 */
7.           if (my_virtual_id AND mask) = 0 then
8.               if (my_virtual_id AND 2^i) = 0 then
9.                   virtual_dest := my_virtual_id XOR 2^i;
10.                  send X to (virtual_dest XOR source);
         /* Convert virtual_dest to the label of the physical destination */
11.              else
12.                  virtual_source := my_virtual_id XOR 2^i;
13.                  receive X from (virtual_source XOR source);
         /* Convert virtual_source to the label of the physical source */
14.              endelse;
15.      endfor;
16.  end GENERAL_ONE_TO_ALL_BC
```

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

Reduction Algorithm

(3/3)

```
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
             /* Select nodes whose lower i bits are 0 */
6.           if (my_id AND mask) = 0 then
7.               if (my_id AND 2^i) ≠ 0 then
8.                   msg_destination := my_id XOR 2^i;
9.                   send sum to msg_destination;
10.              else
11.                  msg_source := my_id XOR 2^i;
12.                  receive X from msg_source;
13.                  for j := 0 to m − 1 do
14.                      sum[j] := sum[j] + X[j];
15.              endelse;
16.          mask := mask XOR 2^i;   /* Set bit i of mask to 1 */
17.      endfor;
18.  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

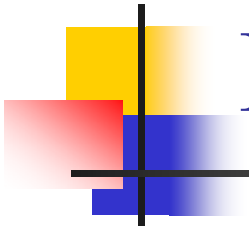
4.1.6 Cost Analysis

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

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

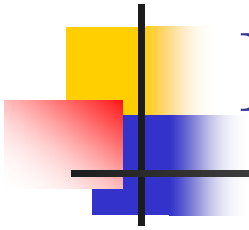
Note: Valid for all topologies

4.2 All-to-All Broadcast and Reduction (1/4)



- All-to-all broadcast can be viewed as a generalization of one-to-all broadcast
- All *p* nodes simultaneously initiate a broadcast
- Each node sends the same *m-word* message to every other nodes
- Different node may broadcast different messages
- The dual of all-to-all broadcast is all-to-all reduction
- Every node is the destination of an all-to-one reduction
- Also known multi node broadcast
- Applications
  - Matrix-Vector Multiplication
  - Matrix-Matrix Multiplication

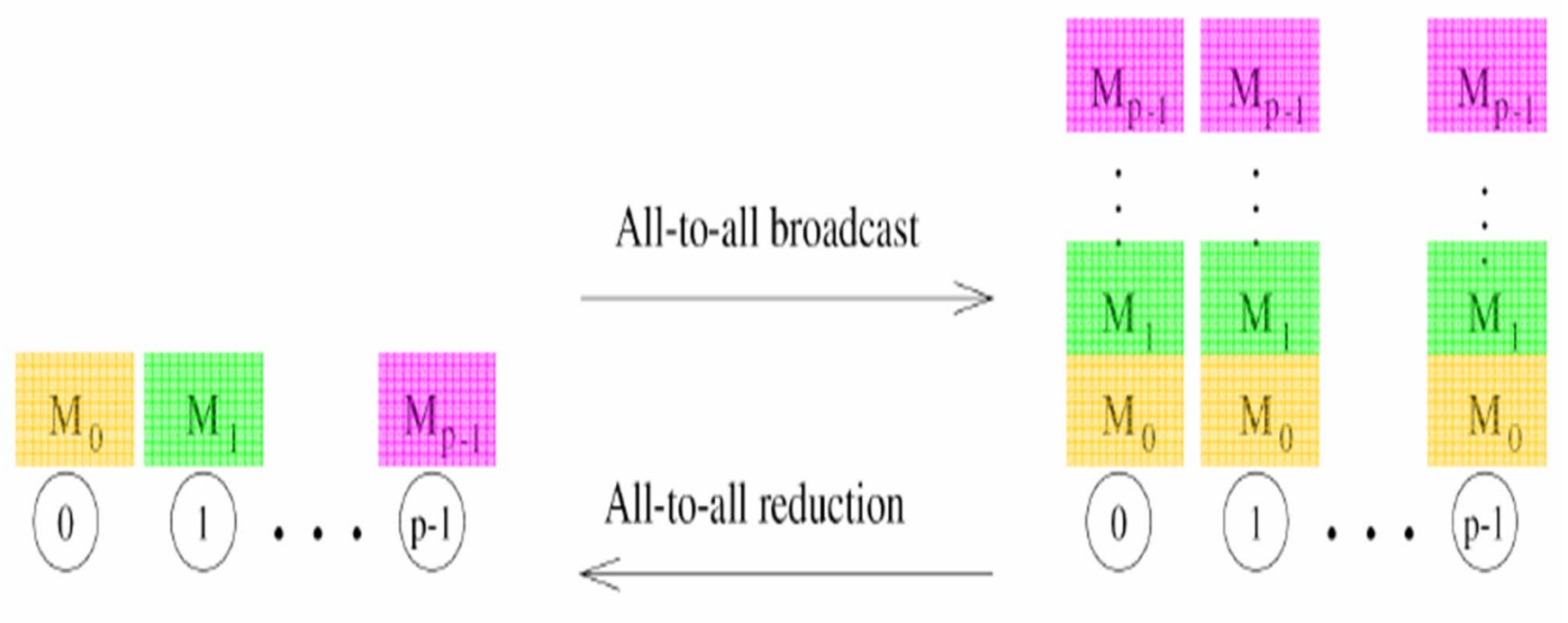
All-to-All Broadcast and Reduction (2/4)



- Naïve approach:
  - sequentially execute p broadcasts/reductions
- Better approach:
  - pipeline the broadcasts
  - be careful not to overload/congest the links

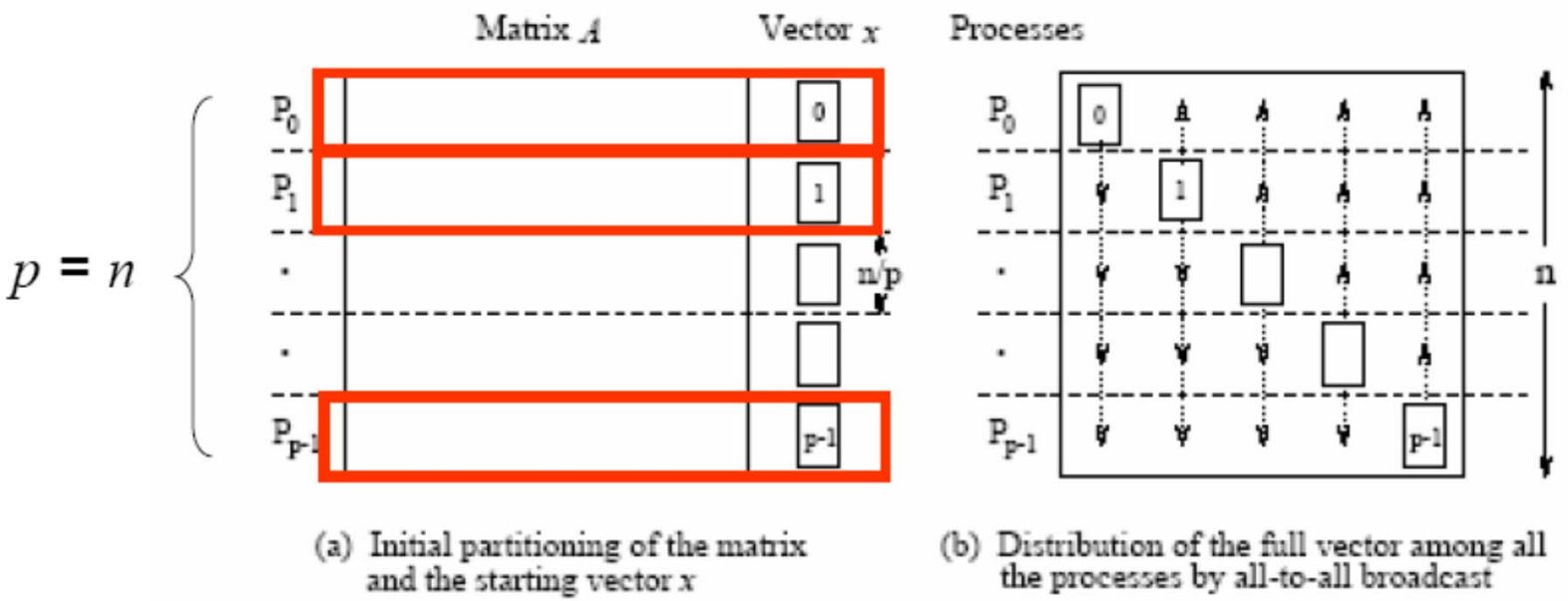
All-to-All Broadcast and Reduction (3/4)

Ref 06-MVP06-slides.pdf



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

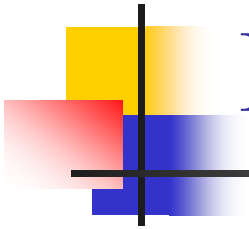
Example: All-to-All Broadcast (4/4)



cz4102_le11.pdf

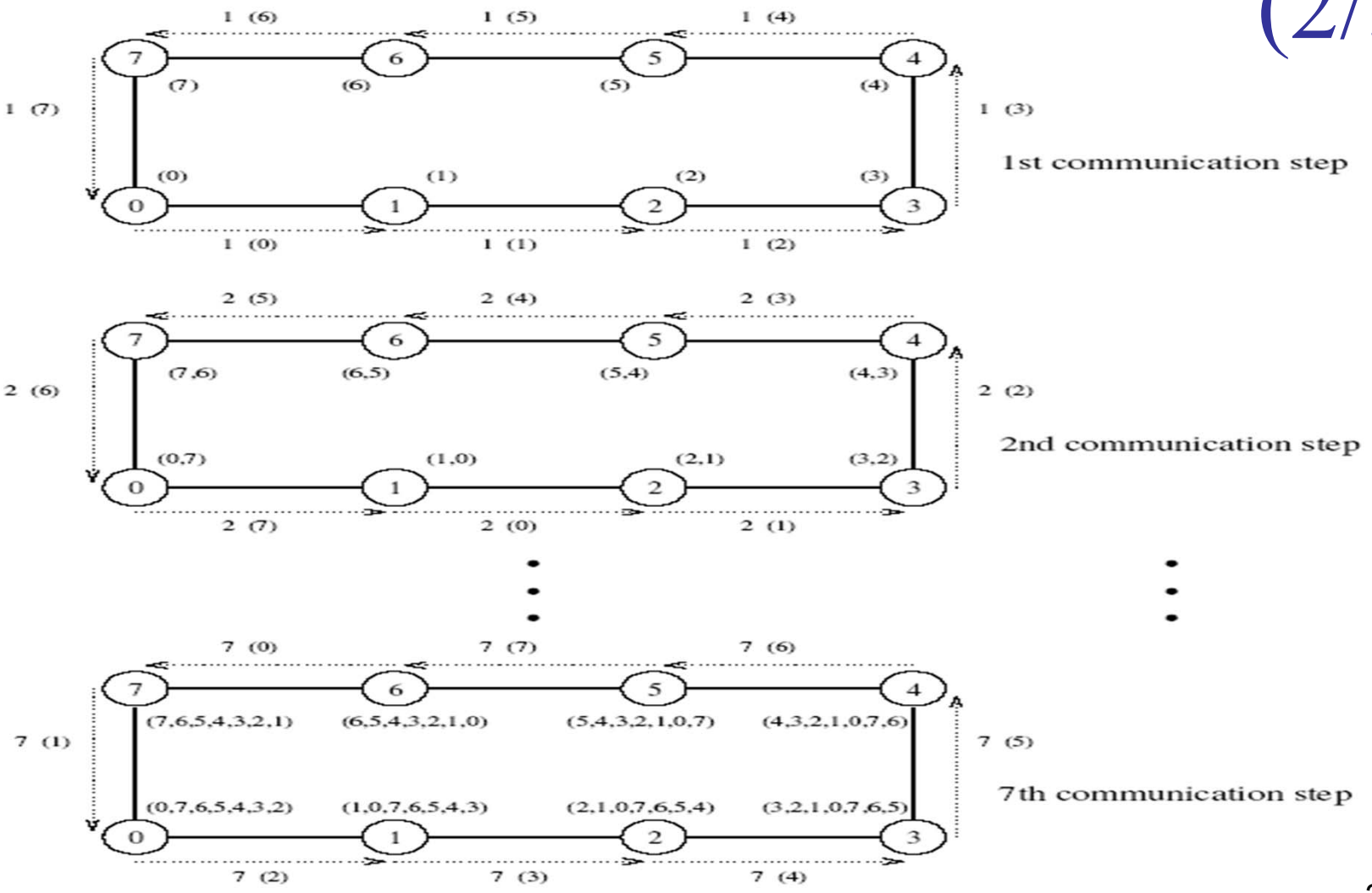
Matrix-Vector Multiplication: Rowwise 1-D Partitioning (p = n).

4.2.1.1 All-to-All Broadcast on a Ring (1/3)



- All-to-all broadcast is achieved by a pipelined point-to-point nearest neighbor communication
- For linear array, bi-directional link is necessary
- Step one:
  - node i sends its data to node (i+1)mod p
- Step 2 .. (p-1)
  - node i sends the data it received in the previous step to node (i+1)mod p

All-to-All Broadcast on a Ring (2/3)



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

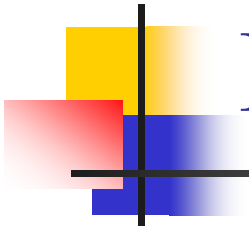
All-to-All Broadcast on a Ring

```
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
```

(3/3)

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

4.2.1.2 All-to-All Reduction on a Ring (1/2)



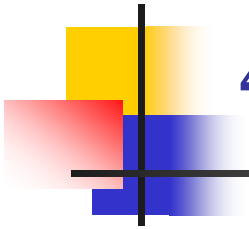
- For all-to-all reduction, the procedure is reversed, each node needs to perform the operation at each step
- The only additional step required is that upon receiving a message, a node must combine it with the local copy that has the same destination
- As the received message before forwarding the combined message

All-to-All Reduction on a Ring

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

(2/2)

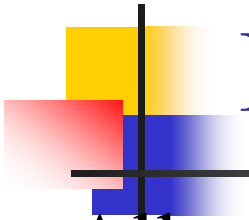
All-to-all broadcast on a p-node ring.



4.2.4.1Cost Analysis for Ring

- On a ring, the time is given by: $(t_s + t_w m)(p-1)$.

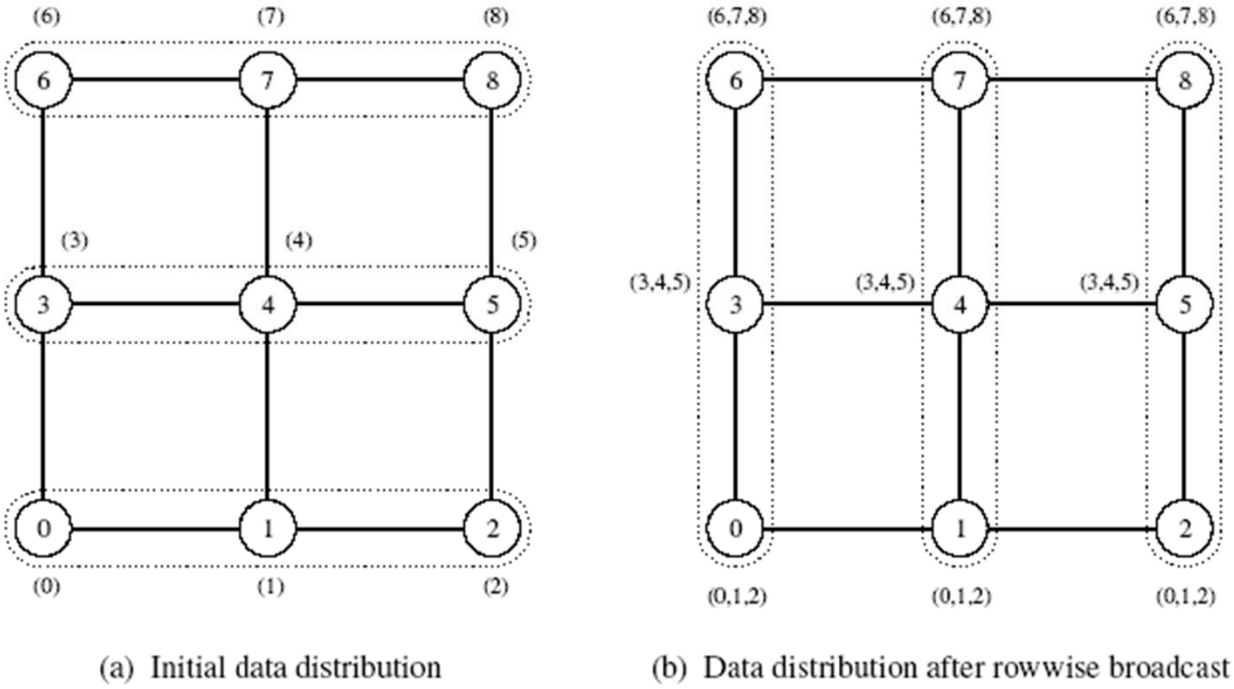
4.2.2 All-to-all Broadcast on a Mesh (1/4)



- All-to-all broadcast algorithm for the 2D mesh is based on the ring algorithm
- The rows and columns of the mesh are treated as rings in two steps:
  - First, each row of the mesh performs an all-to-all broadcast using the ring algorithm, collecting $\sqrt{p}$ messages corresponding to the $\sqrt{p}$ nodes of their respective rows
  - Second, each column performs an all-to-all broadcast, with a single message of size m.$\sqrt{p}$

All-to-all Broadcast on a Mesh

(2/4)



(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 (3/4)

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

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

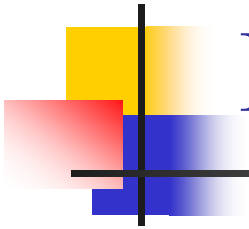
4.2.4.2 Cost Analysis for Mesh (4/4)

- In the first phase, the message size is m, the number of steps is $(\sqrt{p}-1)$
- In the second phase, the message size is m $(\sqrt{p})$, and the number of steps is $(\sqrt{p}-1)$
- The total communication cost is the sum of both phases:

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

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

4.2.3 All-to-all broadcast on a Hypercube (1/4)



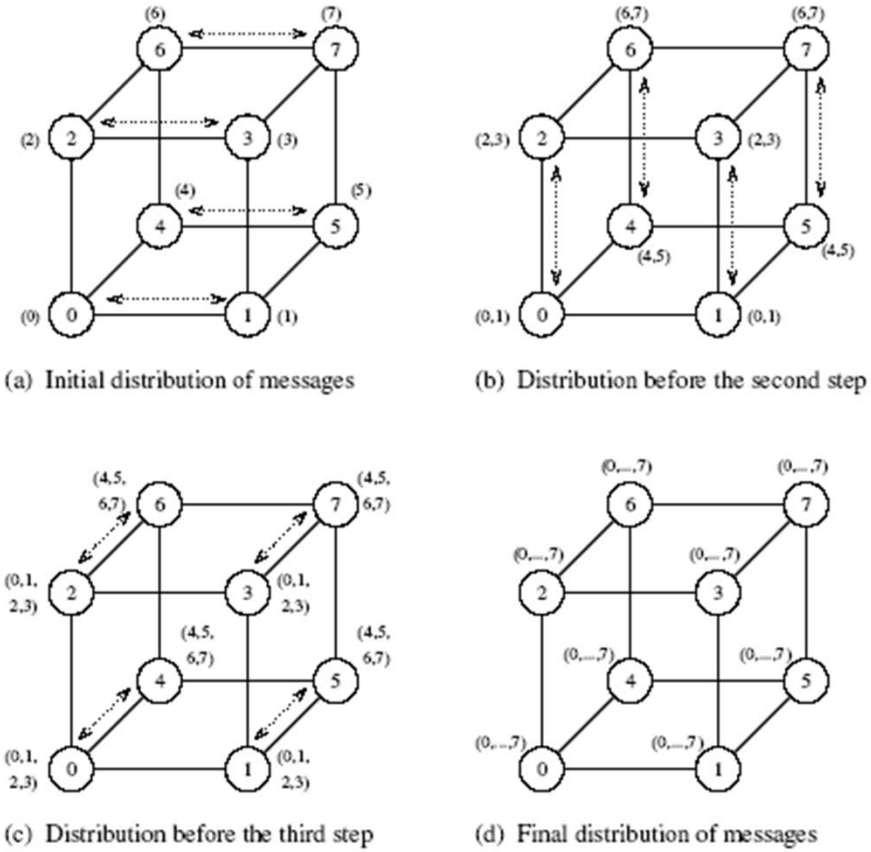
- The all-to-all broadcast needs *log p* steps
- Communication takes place along a different dimension of the hypercube at each step
- At each step, pairs of processors exchange their data
- The size of the message to be transmitted at the next step is doubled by concatenating the received message with their current data

Example: All-to-all broadcast on a 8-node Hypercube (2/4)

- log(p) phases
- phase i: pairs in dimension i exchange data
  data is accumulated

Phase 1	Phase 2	Phase 3
000-001 (0,1) 010-011 (2,3)	000,001-010,011: (0,1,2,3)	000,001,010,011- 100,101,110,111: (0,1,2,3,4,5,6,7)
100-101 (4,5) 110-111 (6,7)	100,101- 110,111: (4,5,6,7)	

Example: All-to-all broadcast on a 8-node Hypercube (3/4)

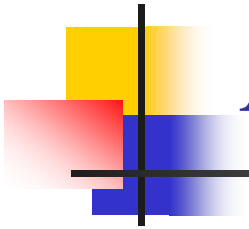


All-to-all broadcast on an eight-node hypercube.

All-to-all broadcast on a Hypercube (4/4)

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

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



All-to-all Reduction (1/2)

- 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.

All-to-all Reduction on a Hypercube (2/2)

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

All-to-all reduction on a d-dimensional hypercube.

4.2.4.3 Cost Analysis for a Hypercube

- *log n* communication rounds
- messages of size $2^{(i-1)}m$ are exchanged in round *i*

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

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

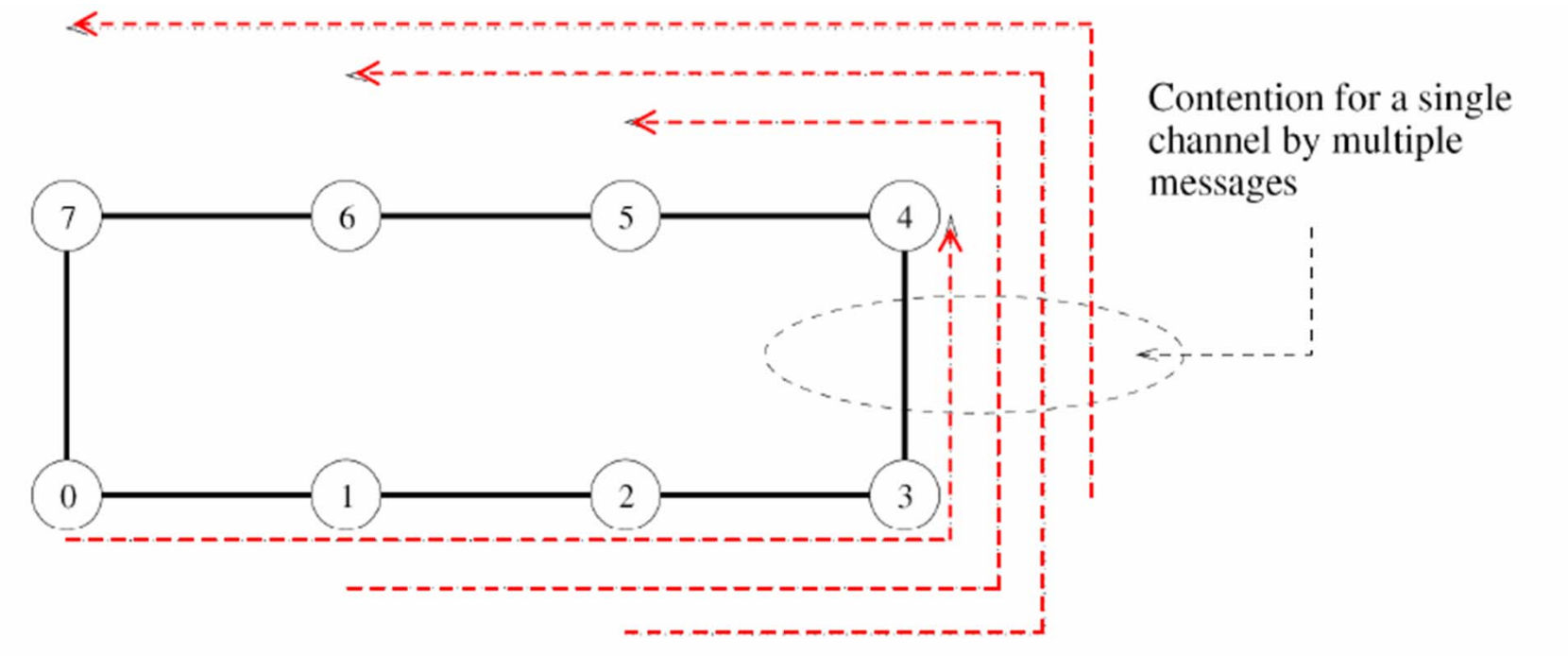
4.2.4.4 All-to-all broadcast: Notes (1/2)

- The bandwidth-limited time $t_w m(p-1)$ is the same for all topologies each processor has to receive total data of $m(p-1)$) bits over a link of bandwidth determined by t_w

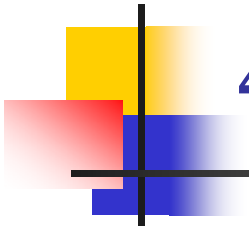
- All of the algorithms presented above are asymptotically optimal in message size.
- It is not possible to port algorithms for higher dimensional networks (such as a hypercube) into a ring because this would cause contention.

Ref 06-MVP06-slides.pdf

All-to-all broadcast: Notes (2/2)



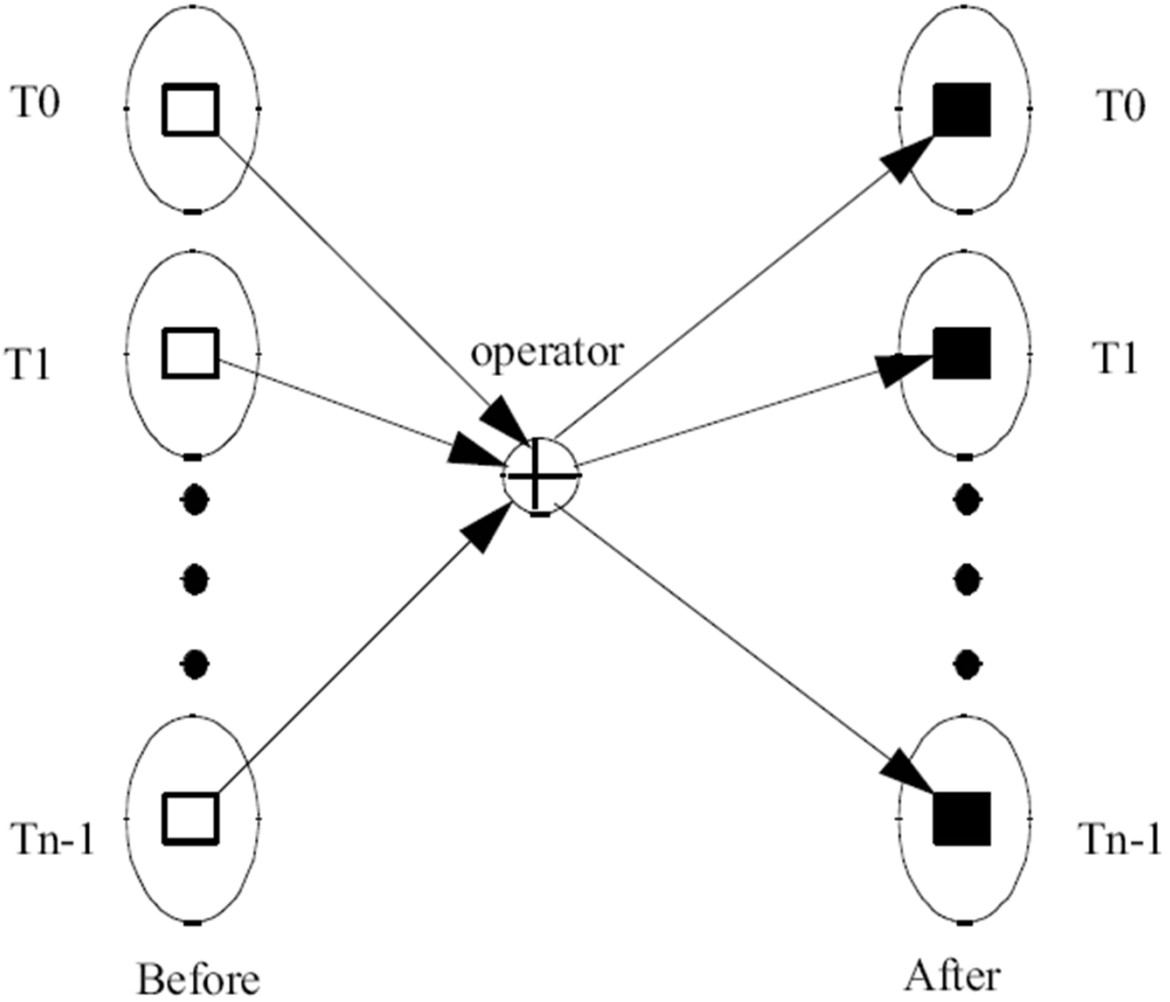
Contention for a channel when the hypercube is mapped onto a ring.



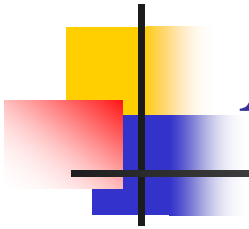
4.3.1 All-Reduce Operation (1/5)

- Identical to *all-to-one* reduction followed by a *one-to-all broadcast*. This formulation is not the most efficient. Uses the pattern of *all-to-all broadcast*, instead. The only difference is that message size does.
- often implements barrier() (synchronization primitive)
- implementation: all-to-all broadcast communication pattern, but instead of concatenating the messages, they are combined according to the operation

All-Reduce Operation (2/5)



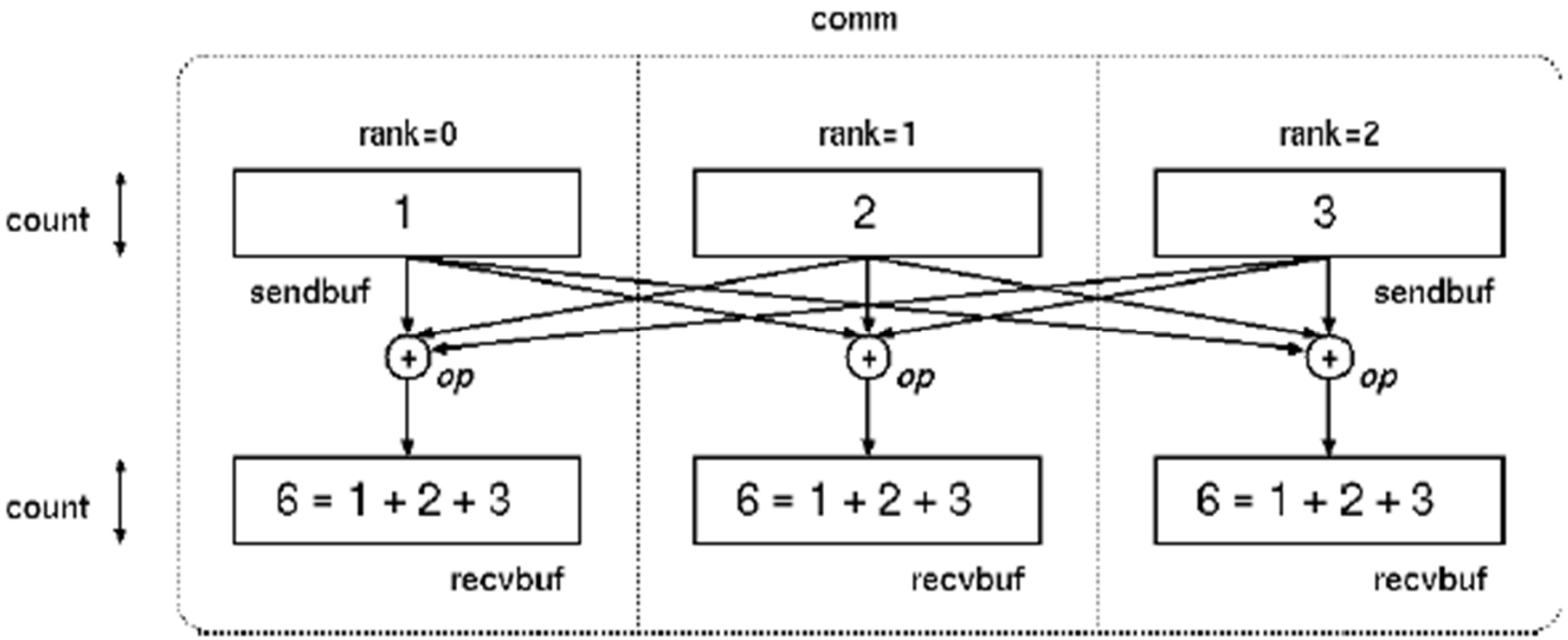
Hesahm,P-222



All-Reduce Operation (3/5)

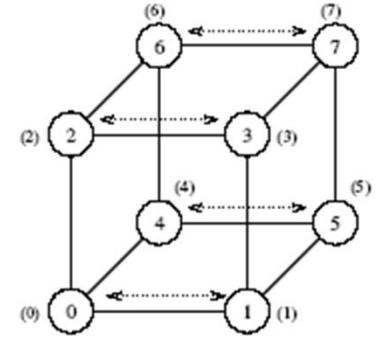
- In all-reduce, each node starts with a buffer of size *m* and the final results of the operation are identical buffers of size *m* on each node that are formed by combining the original *p* buffers using an associative operator.
- Different from all-to-all reduction, in which *p* simultaneous all-to-one reductions take place, each with a different destination for the result.

MPI_ALLREDUCE

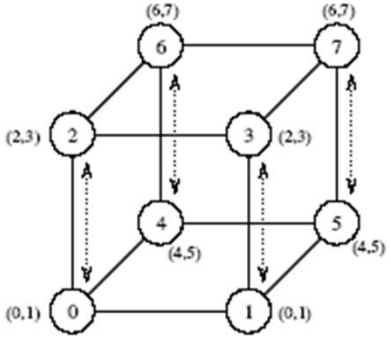


Example: All-Reduce Operation on a 8-node Hypercube (4/5)

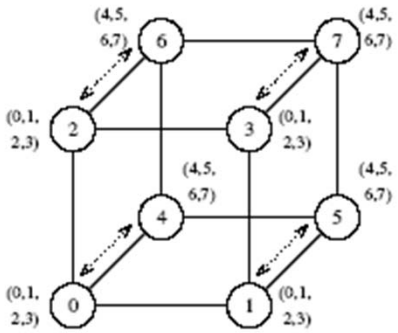
Just replace , by + operation in this figure (4.11)



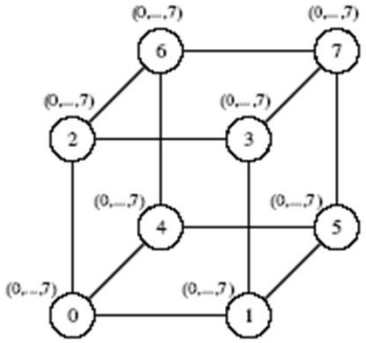
(a) Initial distribution of messages



(b) Distribution before the second step



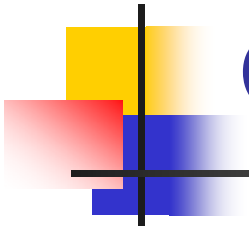
(c) Distribution before the third step



(d) Final distribution of messages

All-Reduce on an eight-node hypercube.

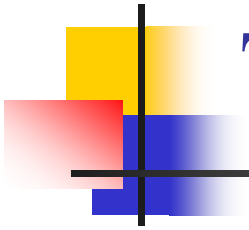
Cost Analysis for All-Reduce Operation using Hypercube (5/5)



- Since the message size does not increase here.
- Time for this operation is $(t_s + t_w m) \log p$.

4.3.2 The Prefix-Sum Operation

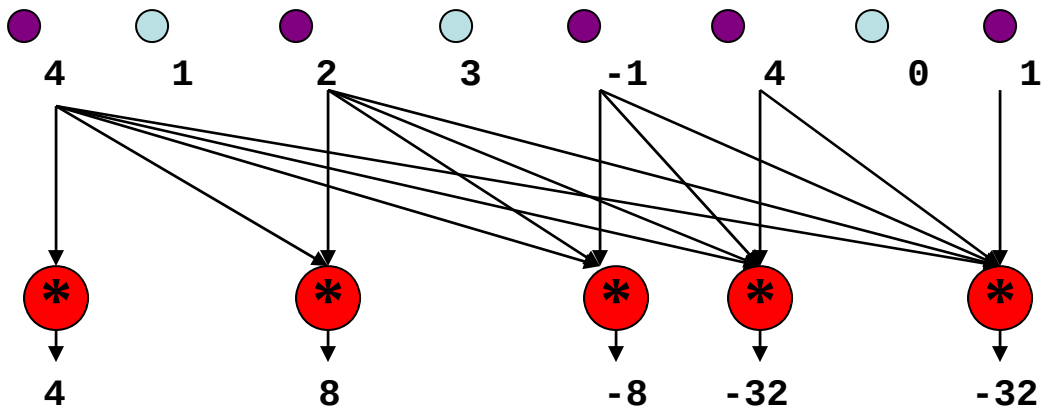
- Also known as scan operation
- Used for associative operations other than addition for finding like maximum, minimum, string concatenations and Boolean operations [Wilkinson, P-172]
- Example:
  - Given p numbers $n_0, n_1, \ldots, n_{p-1}$ (one on each node), the problem is to compute the sums $s_k = \sum_{i=0}^{k} n_i$ 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 (1/4)

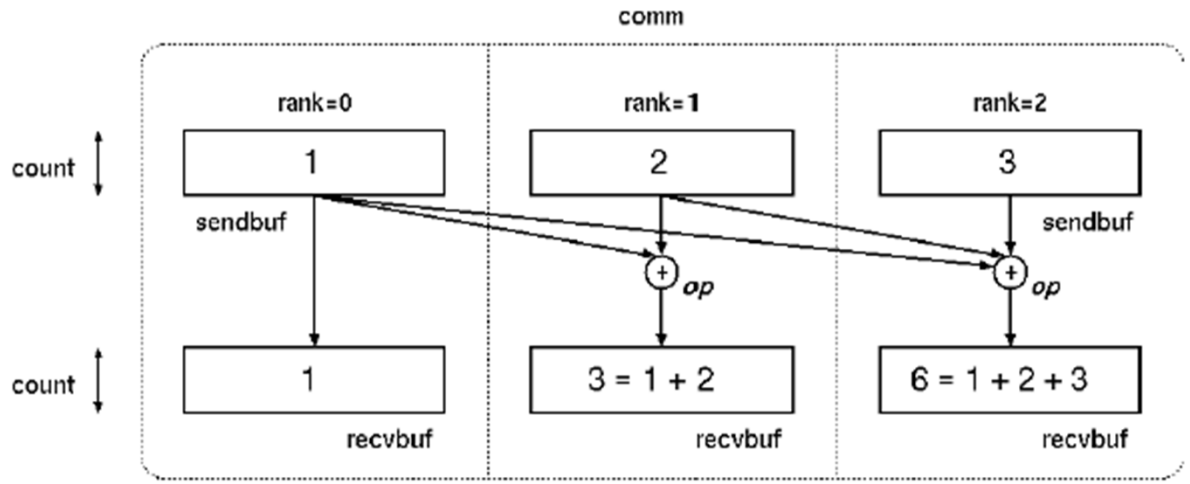
- The operation can be implemented using the all-to-all broadcast kernel.
- We must account for the fact that in prefix sums the node with label k uses information from only the k-node subset whose labels are less than or equal to k.
- This is implemented using an additional result buffer. The content 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.

```
scan(sendBuf, recvBuf,'*', group_id, …);
```

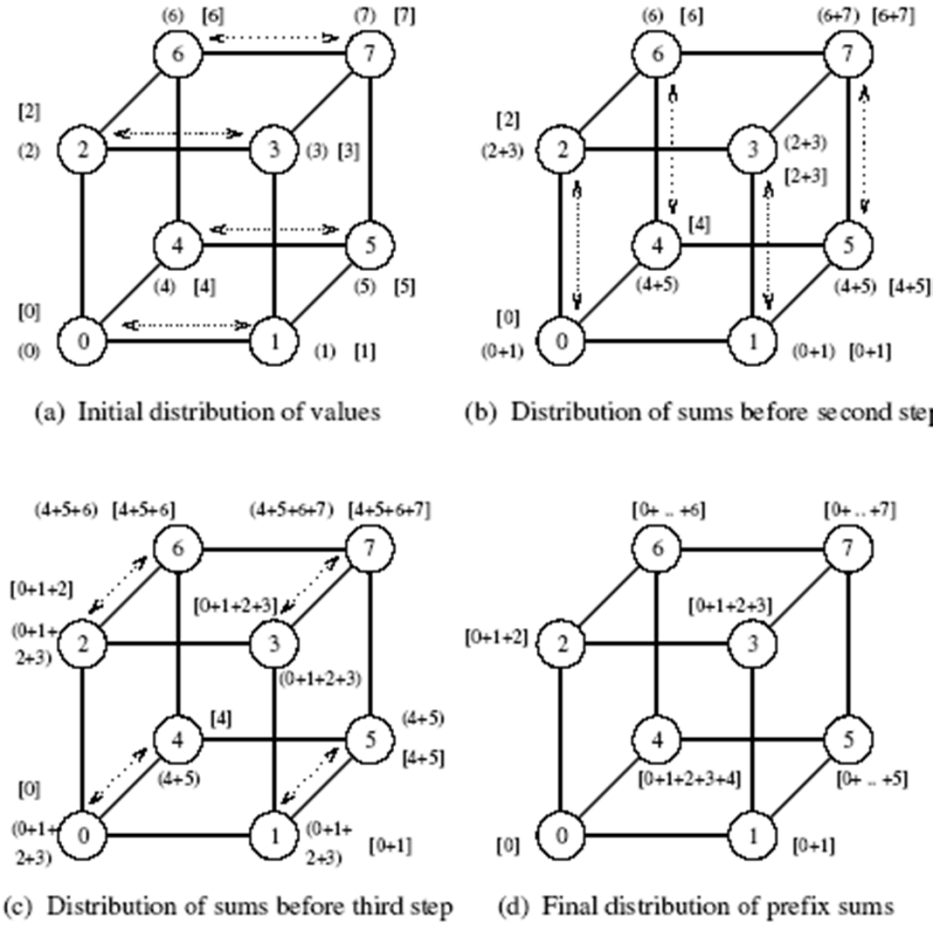


Prefix-Sum

● - group members



The Prefix-Sum Operation (2/4)



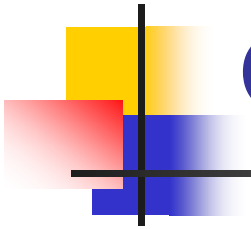
Computing prefix sums on an eight-node hypercube. At each node, square brackets show the local prefix sum accumulated in the result buffer and parentheses enclose the contents of the outgoing message buffer for the next step.

The Prefix-Sum Operation (3/4)

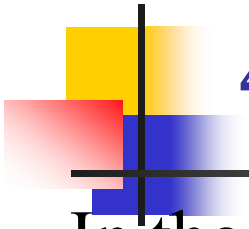
```
1.    procedure PREFIX_SUMS_HCUBE(my_id, my_number, d, result)
2.    begin
3.        result := my_number;
4.        msg := result;
5.        for i := 0 to d − 1 do
6.            partner := my_id XOR 2^i;
7.            send msg to partner;
8.            receive number from partner;
9.            msg := 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.

Cost Analysis for Prefix-Sum Operation using Hypercube (4/4)



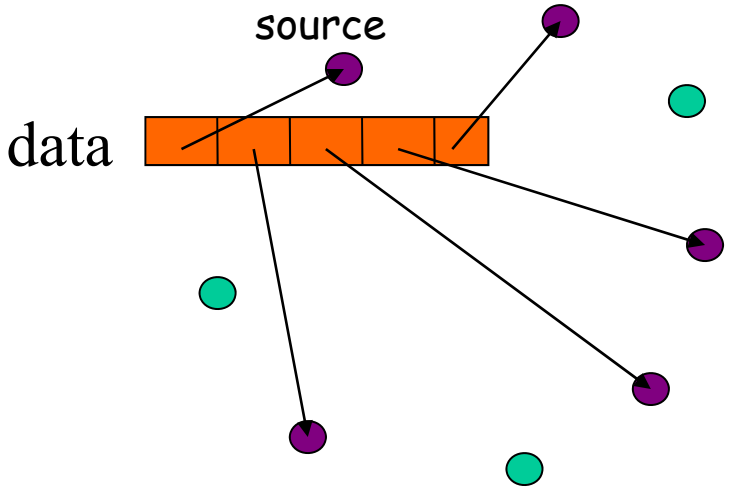
- Some of the messages it receives may be redundant
- Have been omitted in the above algorithm
- The presence or absence of these messages does not affect the results
- Since the message size does not increase here.
- Time for this operation is $(t_s + t_w m) \log p$



4.4 Scatter and Gather (1/4)

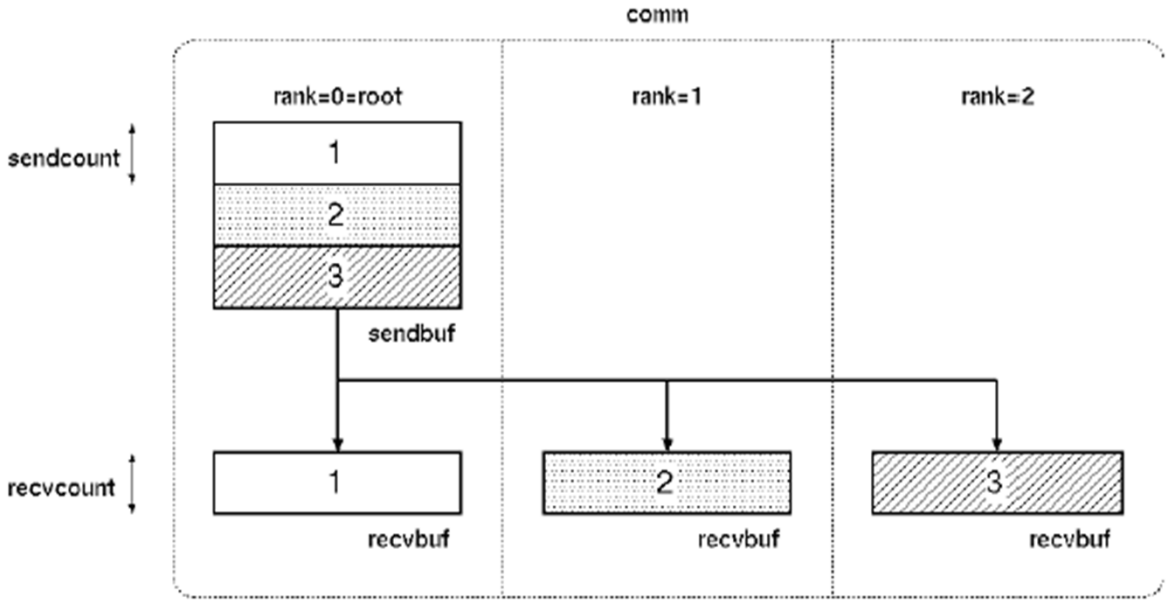
- In the *scatter* operation, a single node sends a unique message of size *m* to every other node (also called a one-to-all personalized communication).
- In the *gather* or *concatenation* operation, a single node collects a unique message from each node.
- While the scatter operation is fundamentally 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.

`scatter(data,…);`

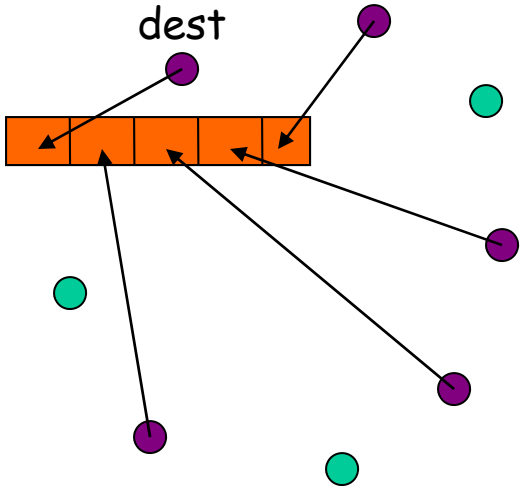


One-to-all personalized communication

● - group members

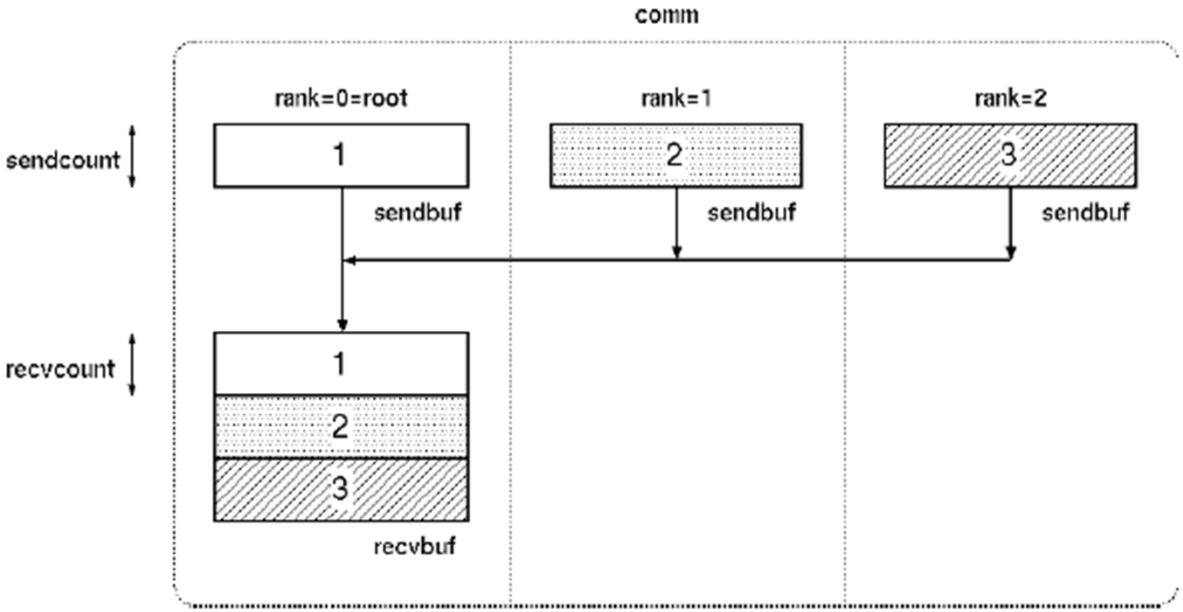


gather(…)

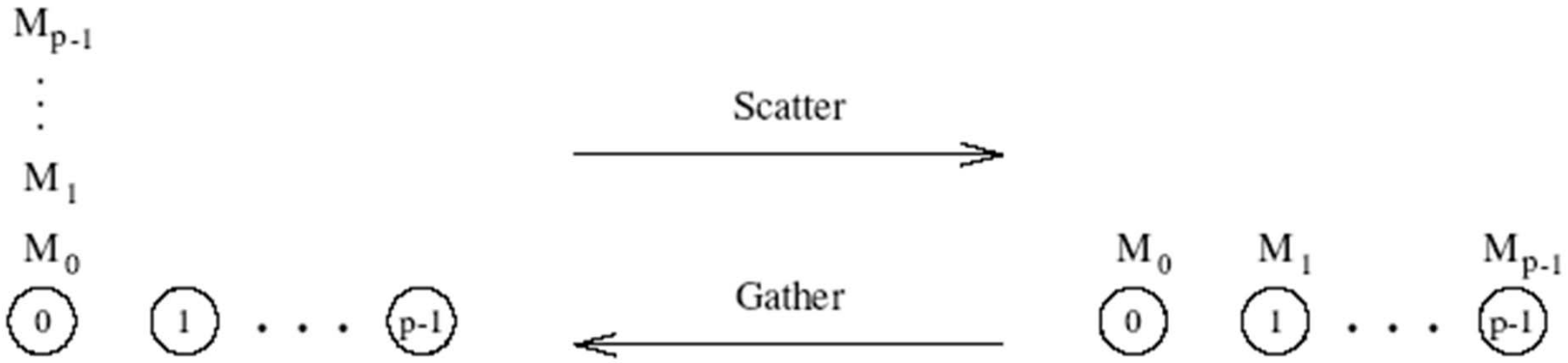


● - group members

concatenation
dual of scatter

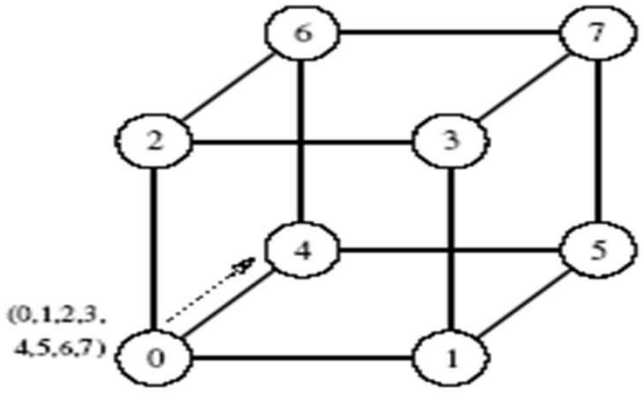


Gather and Scatter Operations (2/4)

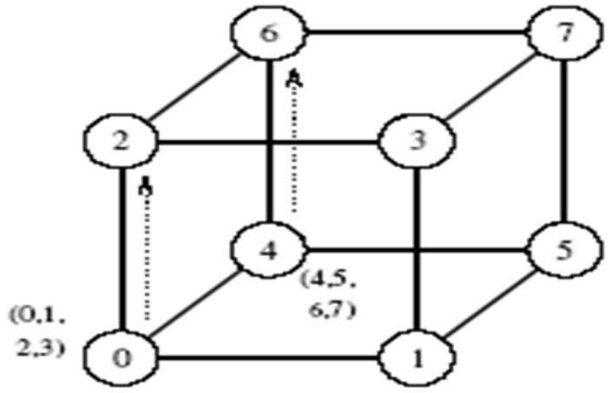


Scatter and gather operations

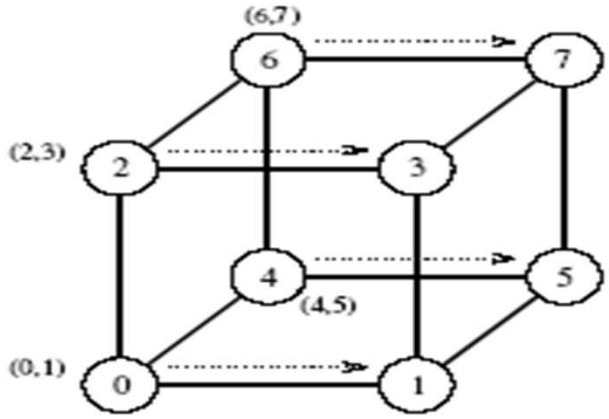
Example of the Scatter Operation (3/4)



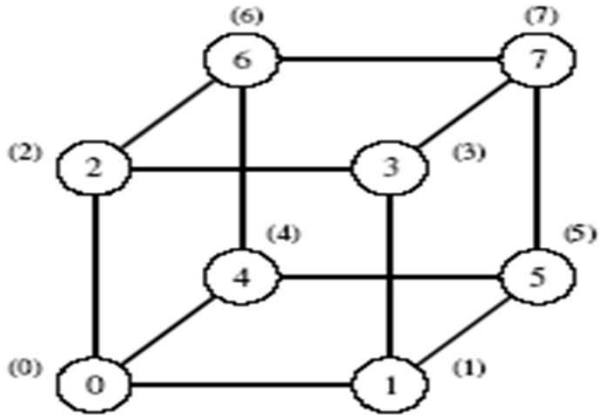
(a) Initial distribution of messages



(b) Distribution before the second step



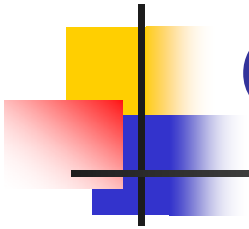
(c) Distribution before the third step



(d) Final distribution of messages

The scatter operation on an eight-node hypercube

Ref 06-MVP06-slides.pdf for explanation



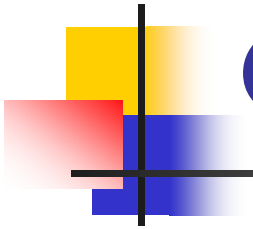
Cost of Scatter and Gather (4/4)

- There are *log p* steps, in each step, the machine size halves and the data size halves.
- We have the time for this operation to be:

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

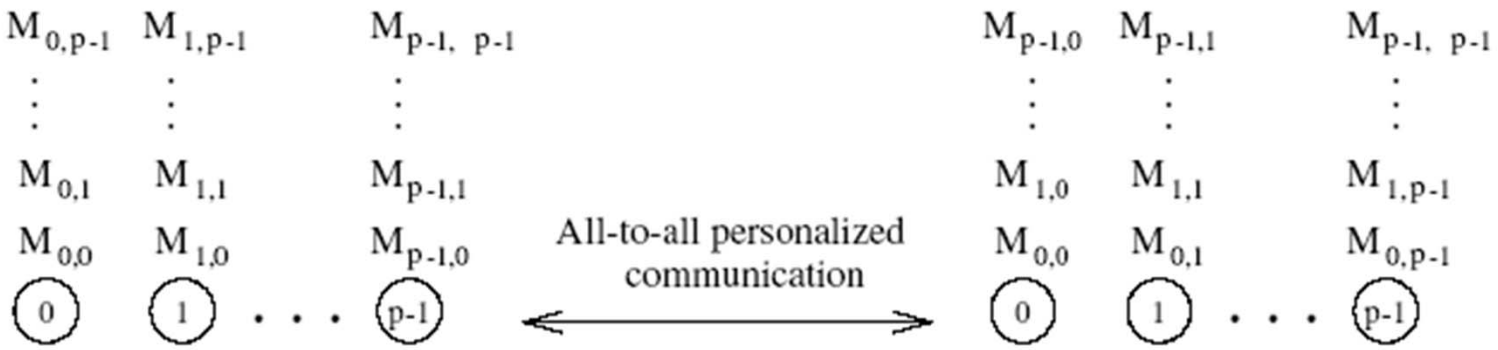
- This time holds for a linear array as well as a 2-D mesh.
- These times are asymptotically optimal in message size.

4.5 All-to-All Personalized Communication (1/2)



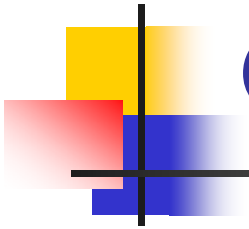
- Each node has a distinct message of size *m* for every other node.
- This is unlike all-to-all broadcast, in which each node sends the same message to all other nodes.
- All-to-all personalized communication is also known as *total exchange*.
- Used in a variety of parallel algorithms such as fast *Fourier transform, matrix transpose, sample sort*, and some *parallel database join* operations

All-to-All Personalized Communication (2/2)



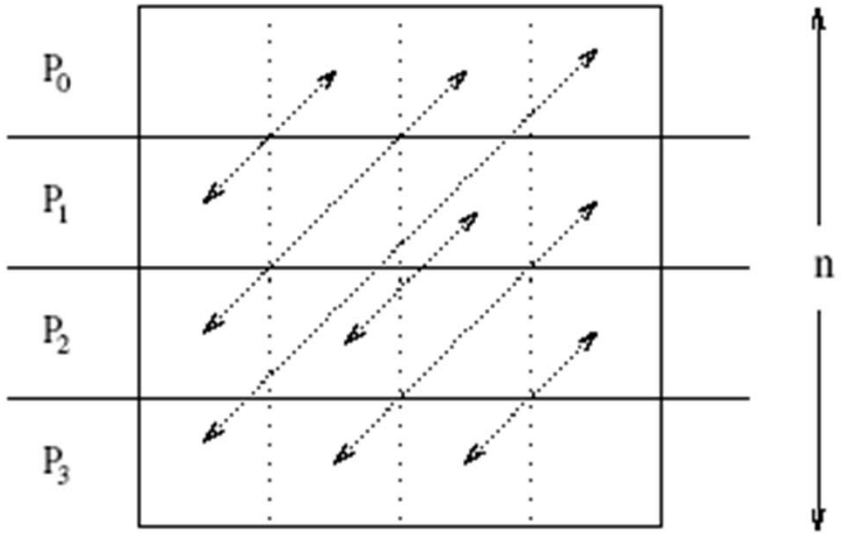
All-to-all personalized communication.

All-to-All Personalized Communication: Example (1/2)



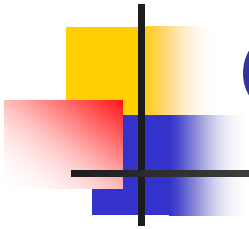
- Consider the problem of transposing a matrix.
- Each processor contains one full row of the matrix.
- The transpose operation in this case is identical to an all-to-all personalized communication operation.

All-to-All Personalized Communication: Example (2/2)



All-to-all personalized communication in transposing a 4 x 4 matrix using four processes.

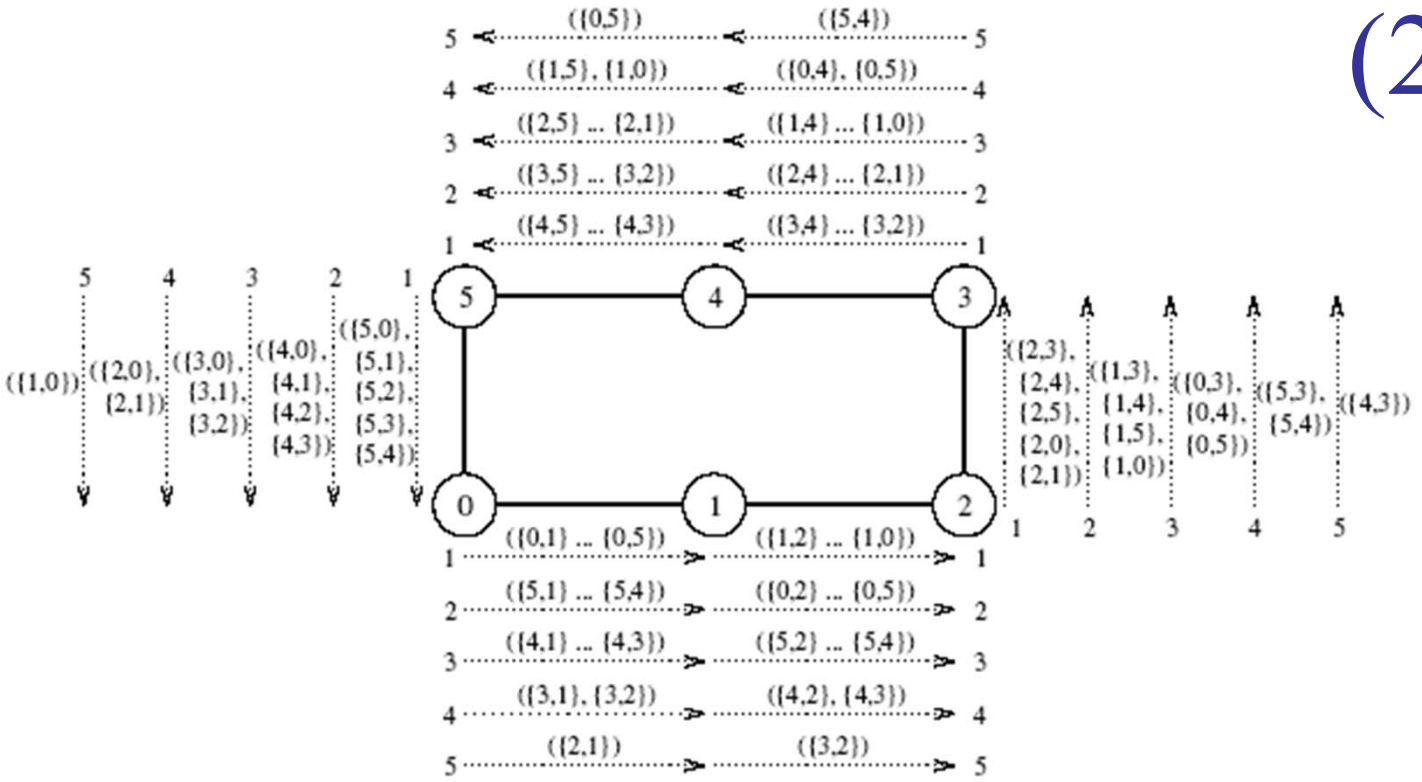
4.5.1 All-to-All Personalized Communication on a Ring (1/3)



- 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

(2/3)



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 ({x1,y1}, {x2,y2},…, {xn,yn}, indicates a message that is formed by concatenating n individual messages.

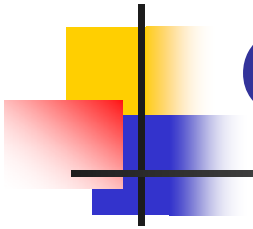
Cost of All-to-All Personalized Communication on a Ring (3/3)

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

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

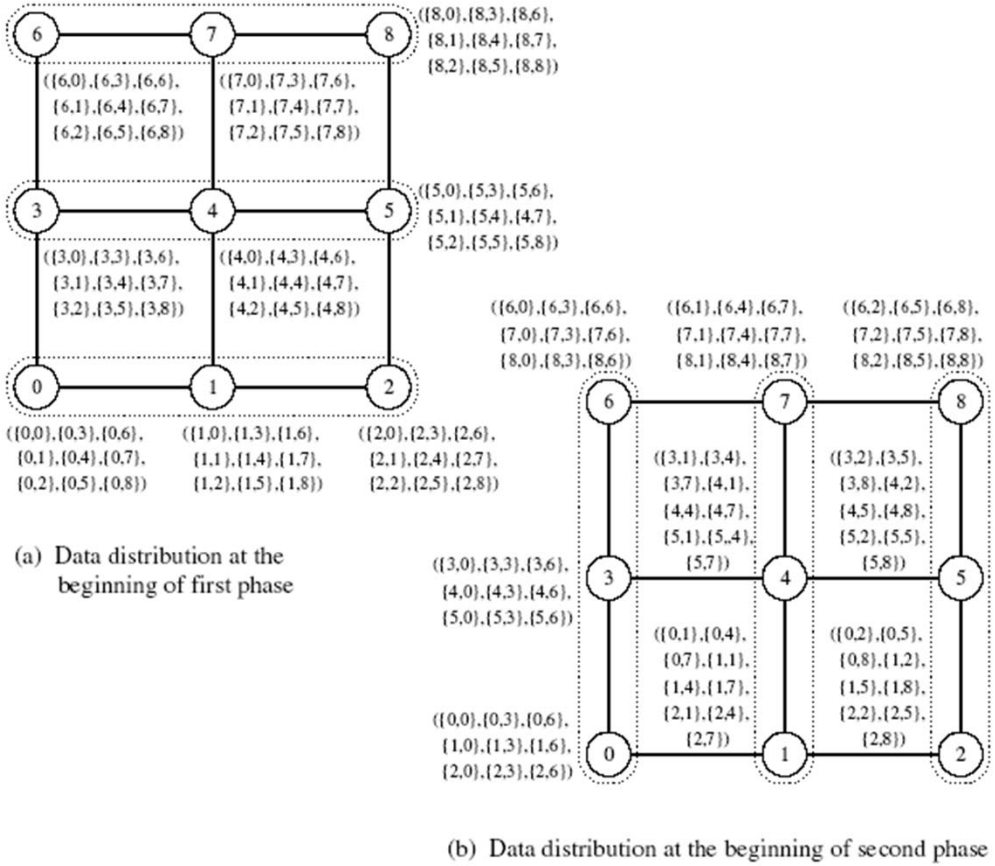
- The t_w term in this equation can be reduced by a factor of 2 by communicating messages in both directions.

4.5.2 All-to-All Personalized Communication on a Mesh (1/3)



- 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 (2/3)



The distribution of messages at the beginning of each phase of all-to-all personalized communication on a 3 x 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.

Cost for All-to-All Personalized Communication on a Mesh (3/3)

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

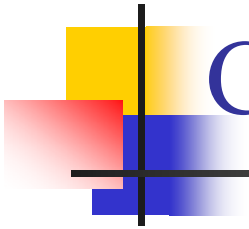
- Time in the second phase is identical to the first phase. Therefore, total time is twice of this time is

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

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

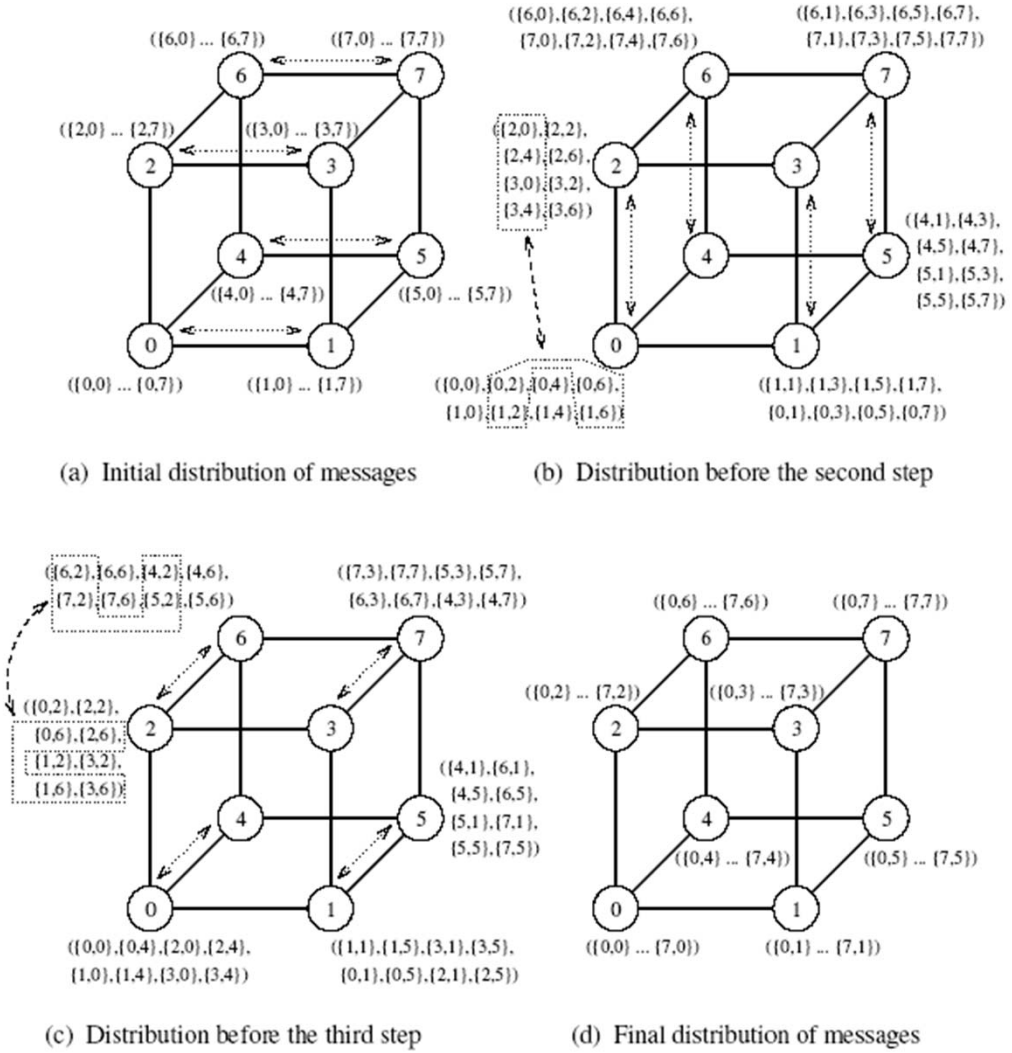
- It can be shown that the time for rearrangement is less much less than this communication time.

4.5.3 All-to-All Personalized Communication on a Hypercube (1/4)



- 4.5.3.1 Naïve Approach
  - 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 using Naïve Approach (2/4)



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

Cost for All-to-All Personalized Communication on Hypercube using Naïve Approach (3/4)

- 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!

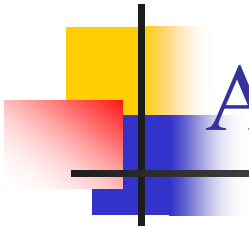
Non-optimality in Hypercube algorithm(4/4)

- Each node send $m(p-1)$ words of data
- Average distance of communication $(\log p)/2$
- Total network traffic is $p \times m(p-1) \times (\log p)/2$
- Total number of links is $(p \log p)/2$
- The lower bound for communication time is

$$T = [t_w p \times m(p-1) \times (\log p)/2] / (p \log p)/2$$

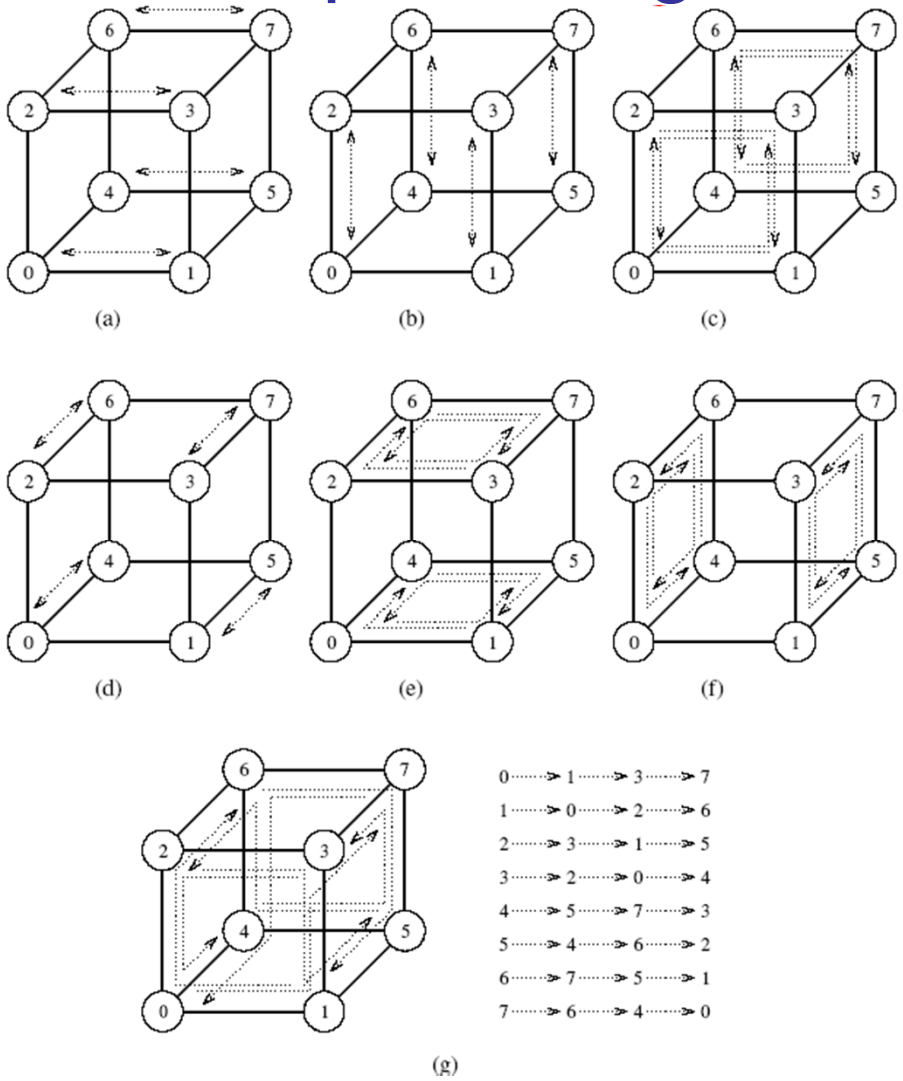
$$= t_w m(p-1)$$

4.5.3.2 All-to-All Personalized Communication on a Hypercube: Optimal Algorithm (1/4)



- 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.
- Processor i communicates with processor $iXORj$ during the jth communication step.
- 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.

All-to-All Personalized Communication on a Hypercube: Optimal Algorithm (2/4)



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

All-to-All Personalized Communication on a Hypercube: Optimal Algorithm (3/4)

```
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
```

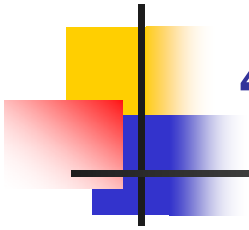
A procedure to perform all-to-all personalized communication on a d-dimensional hypercube. The message $M_{i,j}$ initially resides on node *i* and is destined for node *j*.

All-to-All Personalized Communication on a Hypercube: Cost Analysis of Optimal Algorithm (4/4)

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

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

- This is asymptotically optimal in message size.

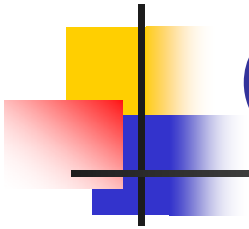


4.6 Circular Shift

- Has application in some matrix computations and in string and image pattern matching
- A special permutation in which node i sends a data packet to node $(i + q)$ mod p in a p-node ensemble

  $(0 \leq q \leq p)$.

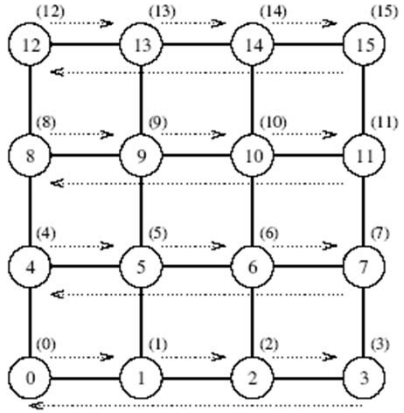
4.6.1 Circular Shift on a Mesh (1/2)



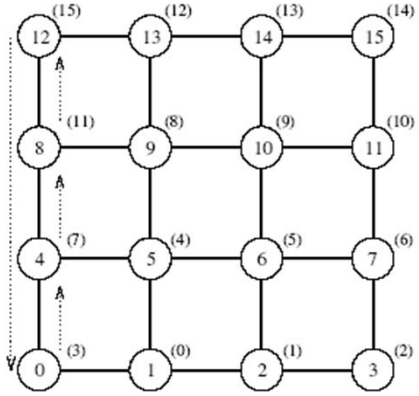
- The implementation on a ring is rather intuitive. It can be performed in min $\{q, p-q\}$ neighbor communications.
- Mesh algorithms follow from this as well. We shift in one direction (all processors) followed by the next direction.
- The associated time has an upper bound of:

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

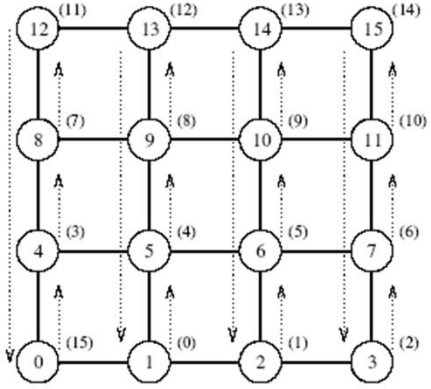
Circular Shift on a Mesh (2/2)



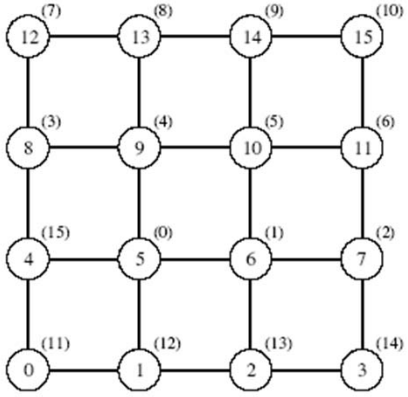
(a) Initial data distribution and the first communication step



(b) Step to compensate for backward row shifts



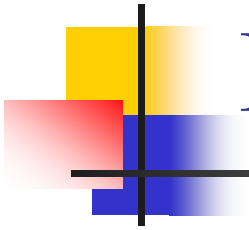
(c) Column shifts in the third communication step



(d) Final distribution of the data

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

4.6.2 Circular Shift on a Hypercube

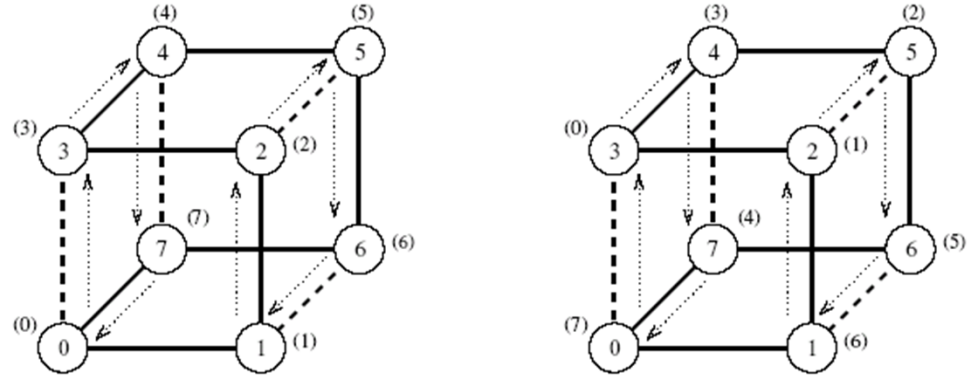


- Map a linear array with 2^d nodes onto a d-dimensional hypercube.
- To perform a q-shift, we expand q as a sum of distinct powers of 2.
- If q is the sum of s distinct powers of 2, then the circular q-shift on a hypercube is performed in s phases.
- The time for this is upper bounded by:

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

- If E-cube routing is used, this time can be reduced to upperbound $T = (t_s + t_w m)(\log p\ ).$

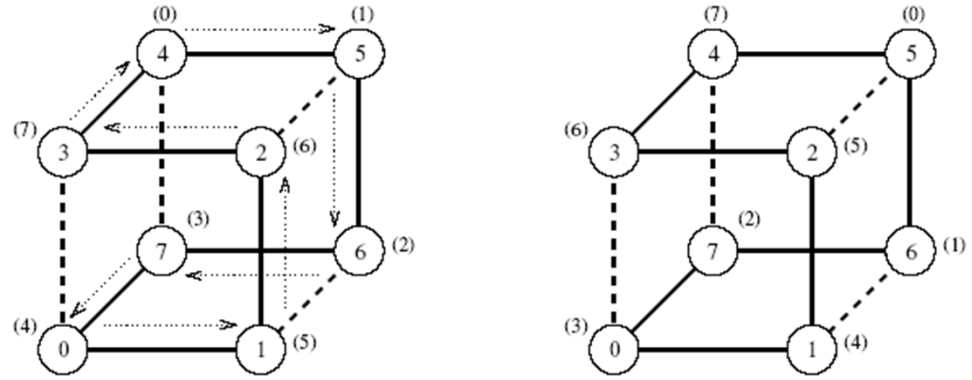
Circular Shift on a Hypercube



First communication step of the 4-shift

Second communication step of the 4-shift

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

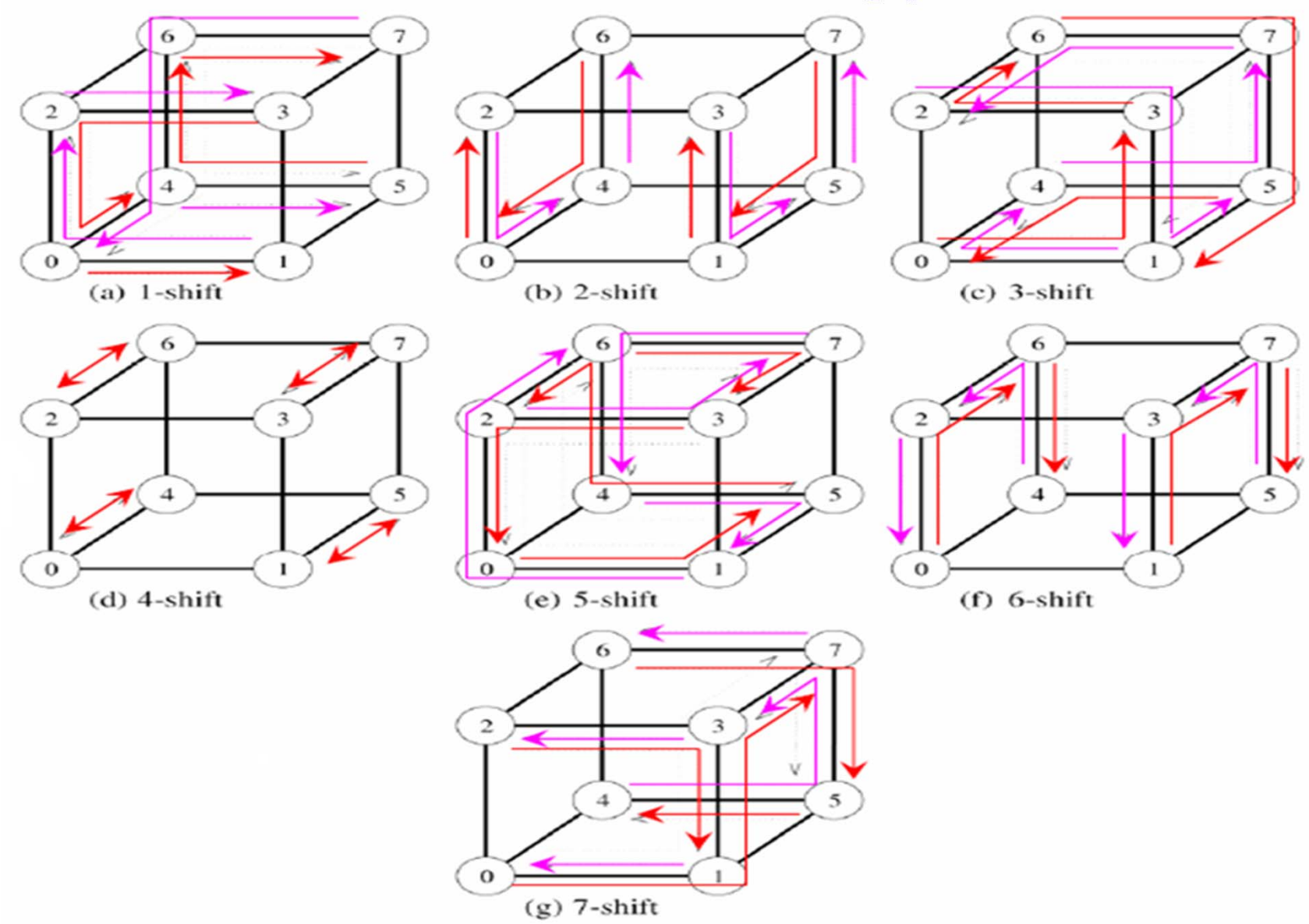


(b) The second phase (a 1-shift)

(c) Final data distribution after the 5-shift

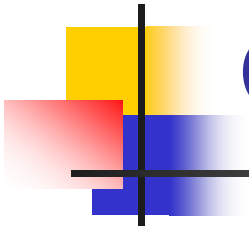
The mapping of an eight-node linear array onto a three-dimensional hypercube to perform a circular 5-shift as a combination of a 4-shift and a 1-shift.

Circular Shift on a Hypercube



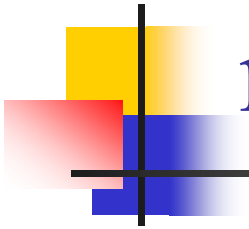
Circular q-shifts on an 8-node hypercube for 1 ≤ q < 8.

4.7 Improving Performance of Operations

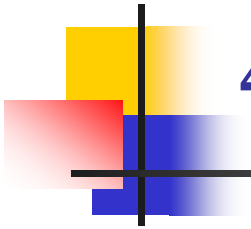


- In all previous derivations in this chapter we assumed that
  - the original messages could not be split into smaller parts and
  - that each node had a single port for sending and receiving data.
- Here we briefly discuss the impact of relaxing these assumptions

4.7.1 Splitting and routing messages into parts

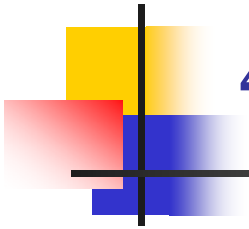


- All discussed operations with exception to one-to-all broadcast, all-to-one reduction and all-reduce are bandwidth optimal
- We can reduce bandwidth complexity of these operations by splitting the message and routing the messages in parts
- m must be big enough
- the factor of t_s increases and the factor of t_w decreases, whether that really speeds up execution depends on t_s, t_w and m



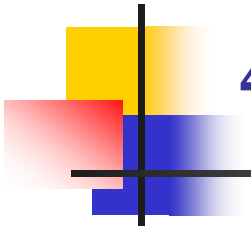
4.7.1.1 One-to-All broadcast

- Scatter *m* into p nodes (fragments of size *m/p*)
- All-to-all broadcast of these fragments, everybody then concatenates the fragments to get original *m*
- Complexity:
  - scatter: $t_s log\ p + t_w(m/p)(p-1)$
  - all-to-all broadcast (hypercube): $t_s log\ p + t_w(m/p)(p-1)$
  - total(hypercube): $2\ (t_s log\ p + t_w m)$ (improvement?)



4.7.1.2 All-to-One Reduction

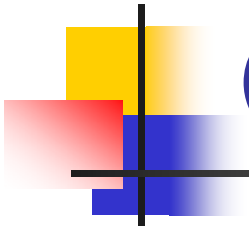
- All-to-one reduction is dual of One-to-All broadcast can be performed
  - All-to-all reduction followed by gather



4.7.1.3 All Reduce

- equals All-to-One reduce followed by One-to-All broadcast
- All-to-One Reduce = All-to-All reduce followed by gather
- One-to-All broadcast = scatter followed by All-to-All broadcast
- inner gather and scatter cancel each other (i.e. the data are already where we wanted them to be)
- = All-to-All reduce followed by All-to-All broadcast
- Complexity= 2 ($t_s log\ p + t_w m$) (improvement?)

4.7.2 All-Port Communication (1/2)



- We assumed single-port communication where as all port communication possible
- On a p-node hypercube with all-port there may be improvement by a factor of *log p*.
- Despite the apparent speedup in several parallel algorithms, an increase in the size of messages means a corresponding increase in the granularity of computation at the nodes

All-Port Communication (2/2)

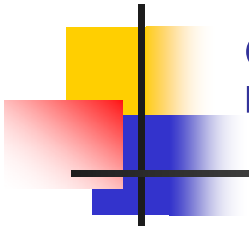
- While working on large data sets, inter-node communication time is dominated by the computation time
  - e.g. for matrix multiplication, n^3 computations for n^2 words of data transferred among the nodes so not much advantageous
- All-port comm. can be effective only if data can be fetched and stored in memory at a rate sufficient to sustain all the parallel communication
  - on a p-node hypercube, the memory b/w must be greater than the communication b/w of a single channel by a factor $\log p$
- Special hardware
  - e.g. for reduction (switches combine messages)

Ref lec4.ppt

Summary (1/3)

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	$t_s + t_w m$	$\Theta(p)$

Summary of communication times of various operations on a hypercube



Summary (2/3)

- They represent regular communication patterns that are performed by parallel algorithms.
  - Collective: Involve groups of processors
- Used extensively in most data-parallel algorithms.
- The parallel efficiency of these algorithms depends on efficient implementation of these operations.
- They are equally applicable to distributed and shared address space architectures
- Most parallel libraries provide functions to perform them
- They are extremely useful for “getting started” in parallel processing!

Summary (2/3)

MPI Names of various Operations

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