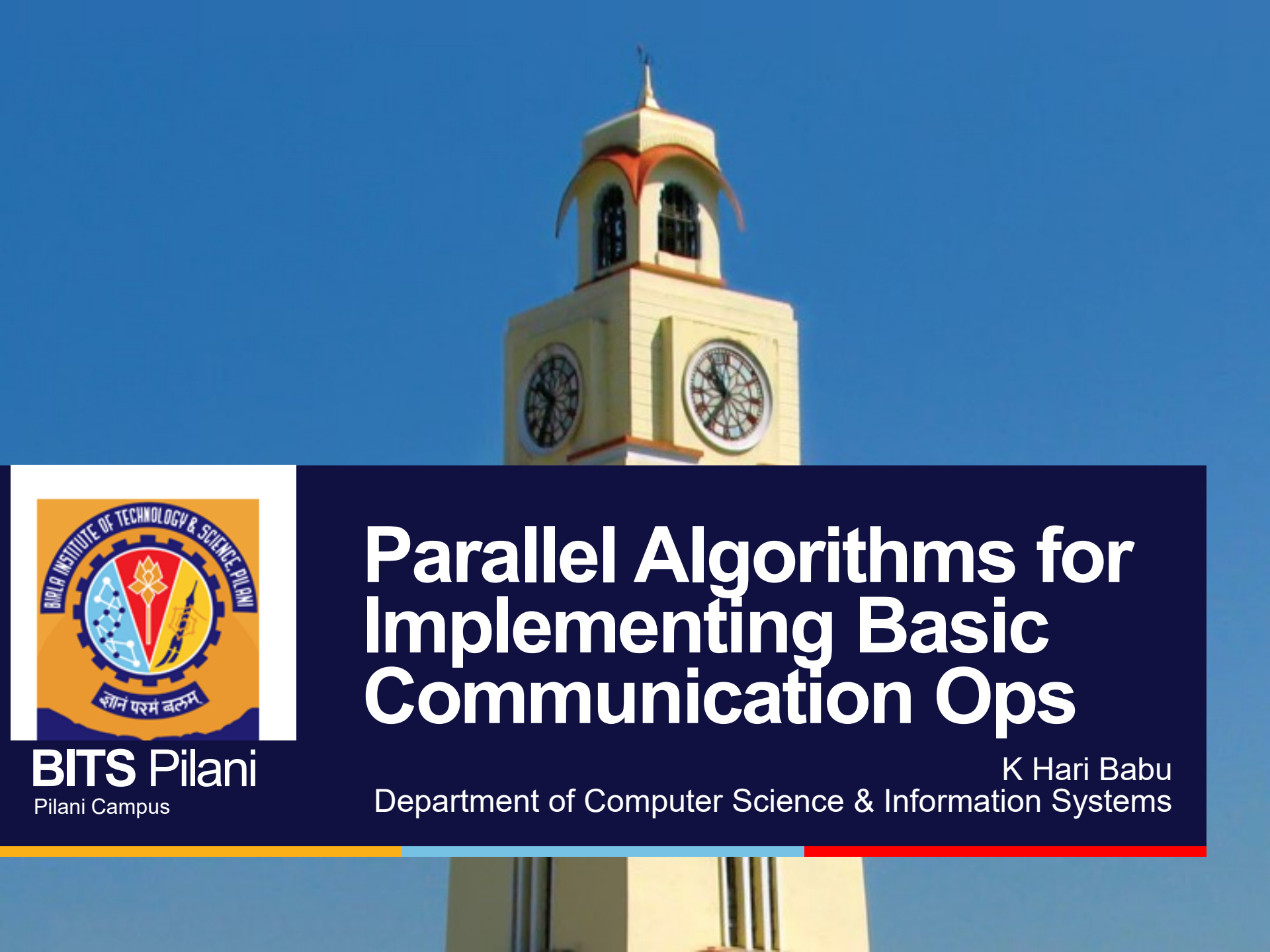


# Parallel Algorithms for Implementing Basic Communication Ops

K Hari Babu
Department of Computer Science & Information Systems

BITS Pilani
Pilani Campus

# Slides Source



- Official slides of section 2.5 and Ch4 of text book are adapted and used.

innovate achieve lead

BITS Pilani
Pilani Campus

# Communication Costs

# Communication Costs

- Sources of overhead in parallel programs
  - Idling
  - Contention
  - Communication costs
    - Major overhead
- Costs depend on
  - Programming model
  - Network topology
  - Routing

# Message Passing Costs

- The time taken to communicate a message between two nodes is the sum of
  - the time to prepare a message for transmission
  - the time taken by the message to traverse the network to its destination
- The principal parameters that determine the communication latency are as follows:
  - Startup time ($t_s$):
    - The startup time is the time required to handle a message at the sending and receiving nodes.
    - This includes the time to prepare the message (adding header, trailer, and error correction information)
    - This delay is incurred only once for a single message transfer.

# Message Passing Costs

- The time taken to communicate a message between two nodes is the sum of
  - the time to prepare a message for transmission
  - the time taken by the message to traverse the network to its destination
- The principal parameters that determine the communication latency are as follows:
  - Per-hop time ($t_h$):
    - This time is a function of number of hops and includes factors such as switch latencies, network delays, etc.
    - The time taken by the header of a message to travel between two directly-connected nodes in the network is called the per-hop time
    - It is also known as node latency
    - The per-hop time is directly related to the latency within the routing switch for determining which output buffer or channel the message should be forwarded to

# Message Passing Costs

- The time taken to communicate a message between two nodes is the sum of
  - the time to prepare a message for transmission
  - the time taken by the message to traverse the network to its destination
- The principal parameters that determine the communication latency are as follows:
  - Per-word transfer time (tw):
    - This time includes all overheads that are determined by the length of the message. This includes bandwidth of links, error checking and correction, etc.
    - If the channel bandwidth is r words per second, then each word takes time $t_w = 1/r$ to traverse the link. This time is called the per-word transfer time. This time includes network as well as buffering overheads.

# Message Passing Costs

- Total time for communication =
  - Startup time ($t_s$)
  - + per-hop time ($t_h$)
  - + per-word transfer time ($t_w$)

# Simplified Cost Model

- The cost of communicating a *m*-word message between two nodes *l* hops away using cut-through routing is given by

$$t_{comm} = t_s + lt_h + t_w m.$$

- In this expression, $t_h$ is typically smaller than $t_s$ and $t_w$. For this reason, the second term in the RHS does not show, particularly, when *m* is large.
- Furthermore, it is often not possible to control routing and placement of tasks.
- For these reasons, we can approximate the cost of message transfer by

$$t_{comm} = t_s + t_w m.$$

innovate achieve lead

BITS Pilani
Pilani Campus

# Basic Communication Operations

# Basic Communication Operations: Introduction

- Many interactions in practical parallel programs occur in well-defined patterns involving groups of processors
  - Either all processes participate together in a single global interaction
  - Or subsets of processes participate in interactions locally
- These interactions are the building blocks in parallel algos
  - 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.
    - Linear array or ring
    - 2D mesh
    - Hypercube

# Basic Communication Operations: Introduction

- These interactions are the building blocks in parallel algos
  - Efficient implementations must leverage underlying architecture. For this reason, we refer to specific architectures here.
    - Linear array or ring
      - Important to understand because rows and cols of 2D mesh are linear arrays
    - 2D mesh
      - Parallel algorithms using regular data structures such as arrays naturally fit into mesh network
    - Hypercube
      - Many algorithms with recursive interaction patterns map naturally onto a hypercube topology

# Basic Communication Operations: Introduction

- Group communication operations are built using point-to-point messaging primitives.
  - On modern parallel computers, time to transfer data between two nodes is independent of the location of two nodes
    - Achieved using firmware and hardware features and cut-through routing, randomized routing algos etc
  - Recall from our discussion of architectures that communicating a message of size $m$ over an uncongested network takes time $t_s + t_m w$.
    - 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
  - Nodes can send messages simultaneously to each other
- Communication is single-ported
  - Node can send a message on only one of its links at a time and node can receive message on only one of its link at a time
  - Sending and receiving simultaneously on same or two different links possible

innovate achieve lead

BITS Pilani
Pilani Campus

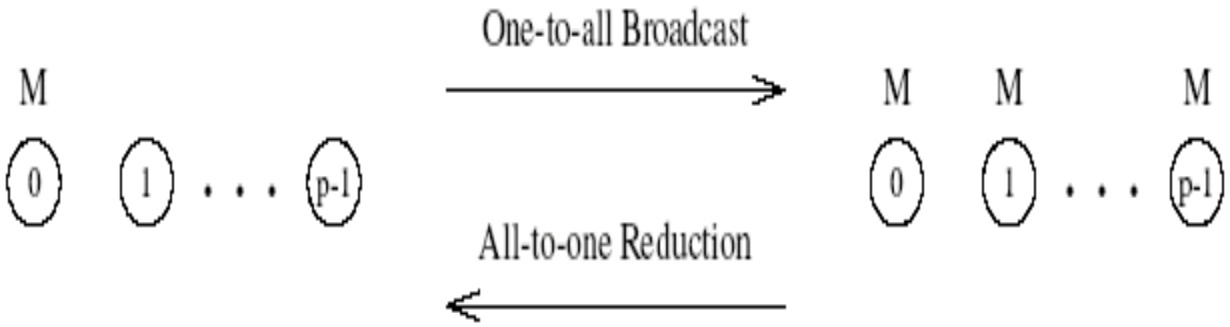
# One-to-All Broadcast and All-to-One Reduction

# 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 processors.



# One-to-All Broadcast on Rings

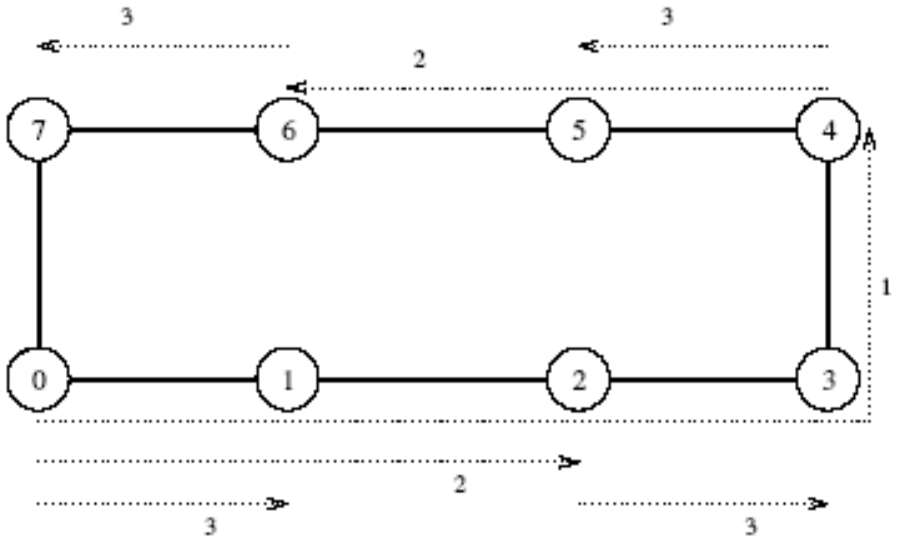
- Simplest way is to send *p-1* messages from the source to the other *p-1* processors
  - this is not very efficient.
  - poor utilization of resources
- Use recursive doubling:
  - source sends a message to a selected process
  - all processes that have the data can send it again.
- Reduction can be performed in an identical fashion by inverting the process.

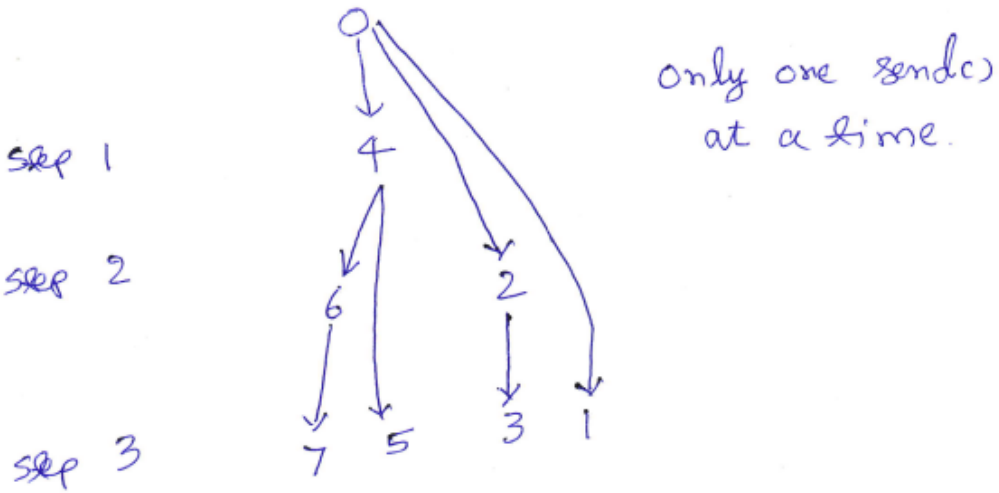
# One-to-All Broadcast

- The source process first sends the message to another process.
- Now both these processes can simultaneously send the message to two other processes that are still waiting for the message.
- By continuing this procedure until all the processes have received the data, the message can be broadcast in **log p** steps

# One-to-All Broadcast

- One-to-all broadcast on an eight-node ring.
  - Node 0 is the source of the broadcast.

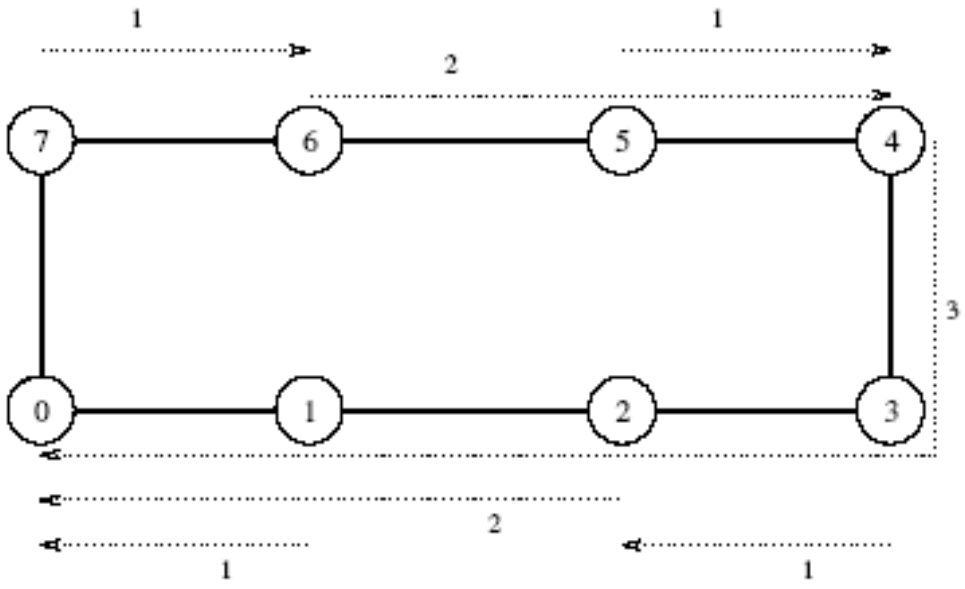



The message is first sent to the farthest node (4) from the source (0). In the second step, the distance between the sending and receiving nodes is halved, and so on. The message recipients are selected in this manner at each step to avoid congestion on the network. For example, if node 0 sent the message to node 1 in the first step and then nodes 0 and 1 attempted to send messages to nodes 2 and 3, respectively, in the second step, the link between nodes 1 and 2 would be congested as it would be a part of the shortest route for both the messages in the second step.

# All-to-One Reduction

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




In the first step, each odd numbered node sends its buffer to the even numbered node just before itself, where the contents of the two buffers are combined into one. After the first step, there are four buffers left to be reduced on nodes 0, 2, 4, and 6, respectively. In the second step, the contents of the buffers on nodes 0 and 2 are accumulated on node 0 and those on nodes 6 and 4 are accumulated on node 4. Finally, node 4 sends its buffer to node 0, which computes the final result of the reduction

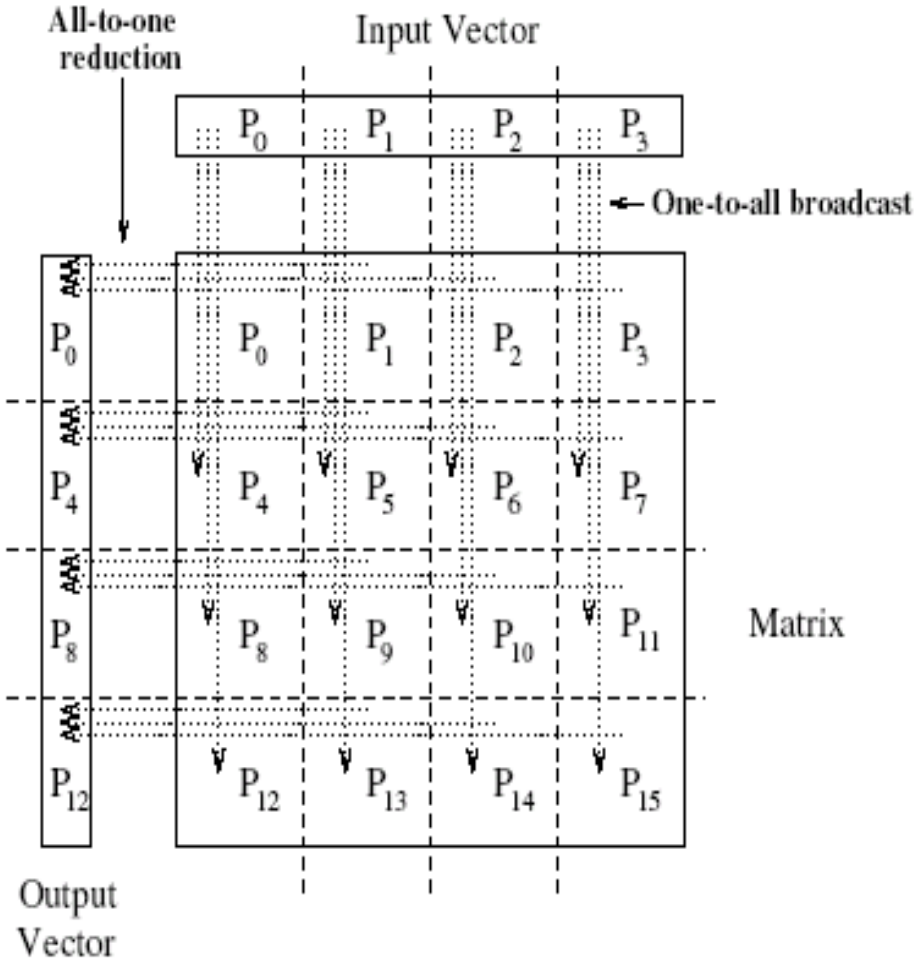
# Broadcast and Reduction: Example

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: Matrix-Vector Multiplication Example

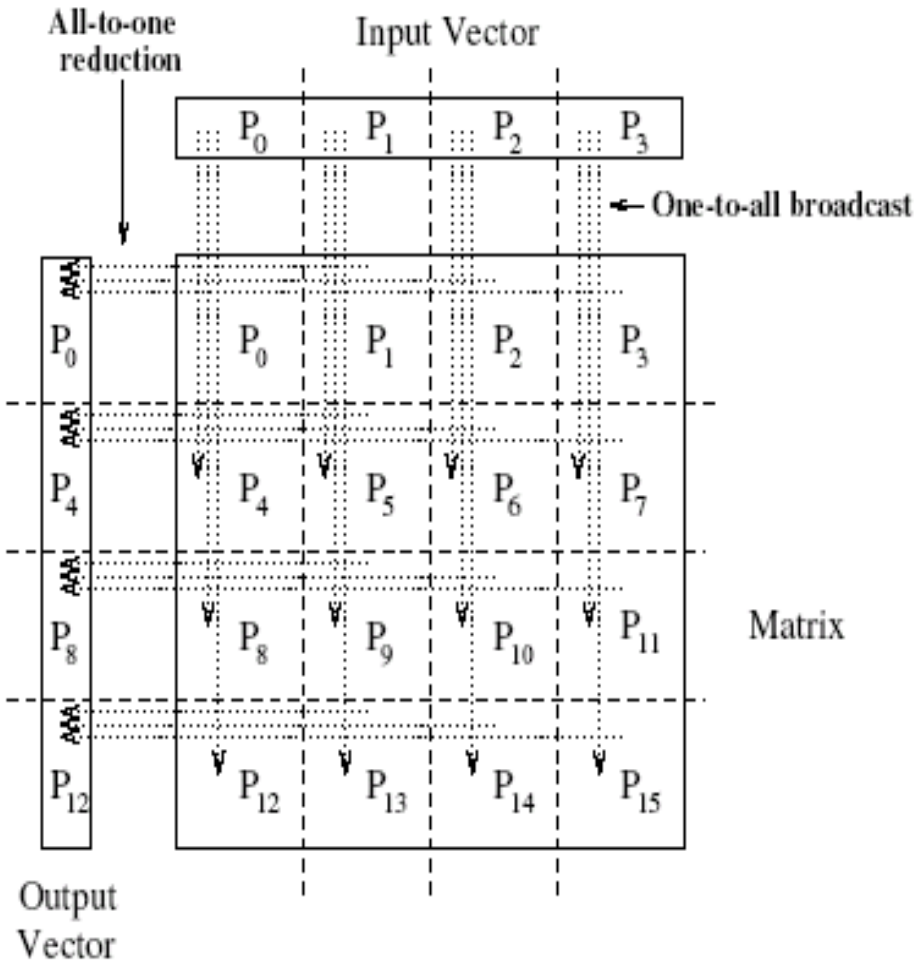
One-to-all broadcast and all-to-one reduction in the multiplication of a *4 x 4* matrix with a *4 x 1* vector.




Since all the rows of the matrix must be multiplied with the vector, each process needs the element of the vector residing in the topmost process of its column. Hence, before computing the matrix-vector product, each column of nodes performs a one-to-all broadcast of the vector elements with the topmost process of the column as the source. This is done by treating each column of the n x n mesh as an n-node linear array, and simultaneously applying the linear array broadcast procedure described previously to all columns.

# Broadcast and Reduction: Matrix-Vector Multiplication Example

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




After the broadcast, each process multiplies its matrix element with the result of the broadcast. Now, each row of processes needs to add its result to generate the corresponding element of the product vector. This is accomplished by performing all-to-one reduction on each row of the process mesh with the first process of each row as the destination of the reduction operation.

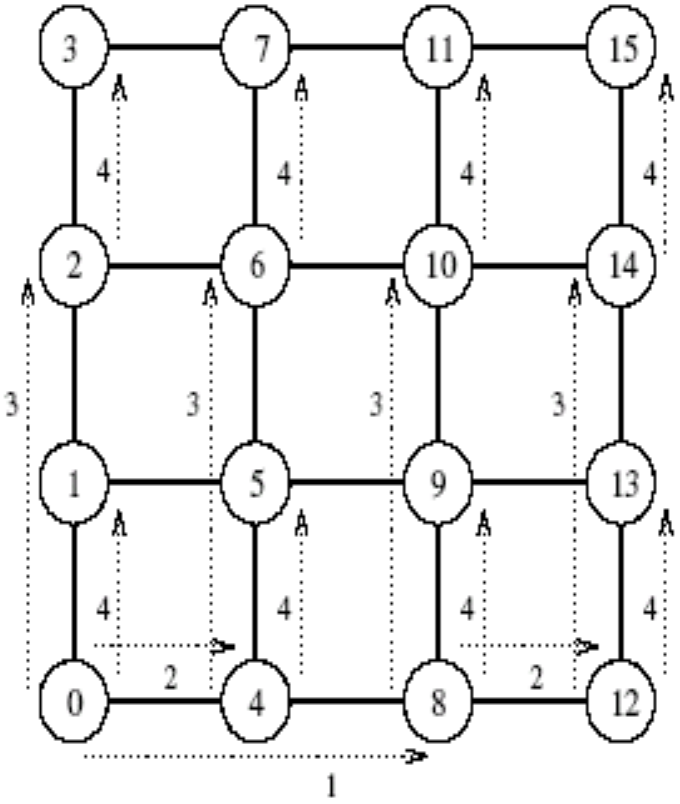
For example, $P_9$ will receive x [1] from $P_1$ as a result of the broadcast, will multiply it with A[2, 1] and will participate in an all-to-one reduction with $P_8$, $P_{10}$, and $P_{11}$ to accumulate y[2] on $P_8$

# 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.

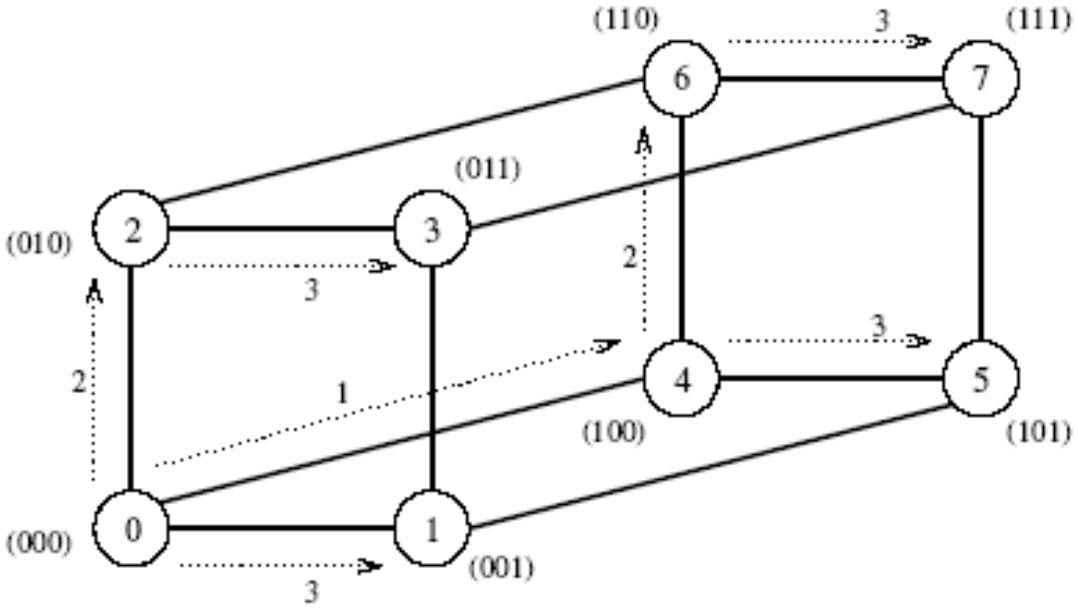



For p = 16, with node 0 at the bottom-left corner as the source. Steps 1 and 2 correspond to the first phase, and steps 3 and 4 correspond to the second phase

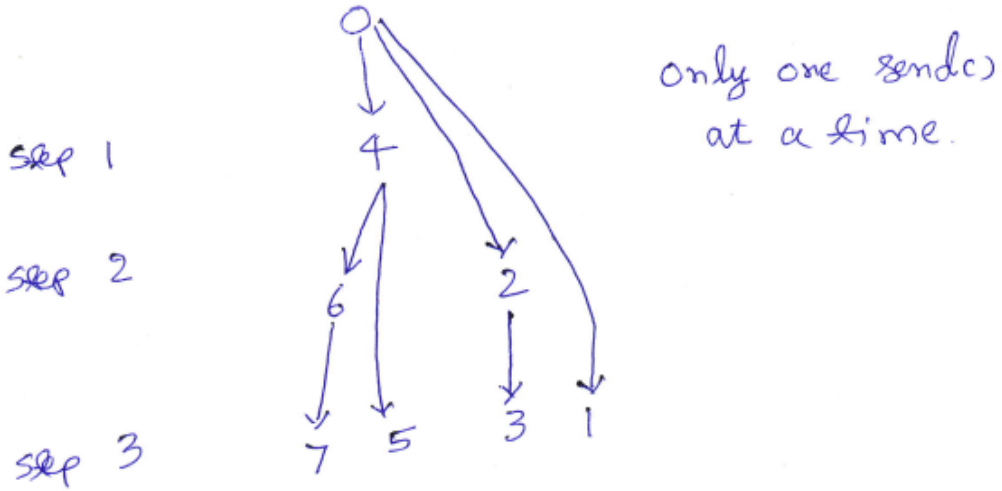
# Broadcast and Reduction on a Hypercube

- One-to-all broadcast is performed in two phases on a two-dimensional mesh, with the communication taking place along a different dimension in each phase
  - Similarly, the process is carried out in three phases on a three-dimensional mesh
  - A hypercube with $2^d$ nodes can be regarded as a d-dimensional mesh with two nodes in each dimension. Hence, the mesh algorithm can be extended to the hypercube, except that the process is now carried out in d steps - one in each dimension
- The mesh algorithm can be generalized to a hypercube and the operation is carried out in $d$ ($= \log p$) steps
  - One step is sufficient in one dimension as there are only two nodes.

# Broadcast and Reduction on a Hypercube: Example



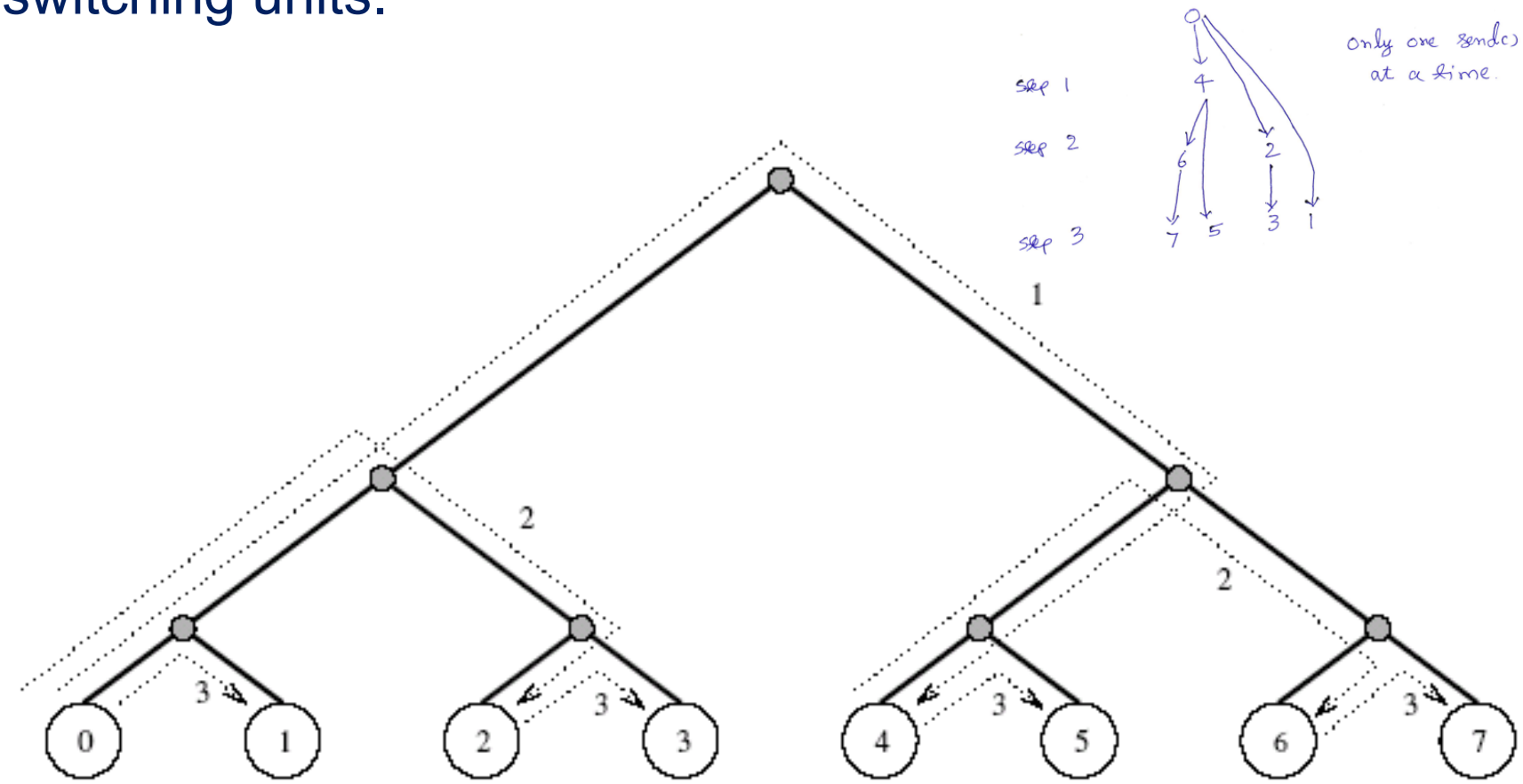

Communication starts along the highest dimension (that is, the dimension specified by the most significant bit of the binary representation of a node label) and proceeds along successively lower dimensions in subsequent steps.




Note that the source and the destination nodes in three communication steps of the algorithm shown above are identical to the ones in the broadcast algorithm on a linear array shown below. However, on a hypercube, the order in which the dimensions are chosen for communication does not affect the outcome of the procedure.

# Broadcast and Reduction on a Balanced Binary Tree

- The hypercube algorithm for one-to-all broadcast maps naturally onto a balanced binary tree in which each leaf is a processing node and intermediate nodes serve only as switching units.



# Broadcast and Reduction Algorithms

- 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.

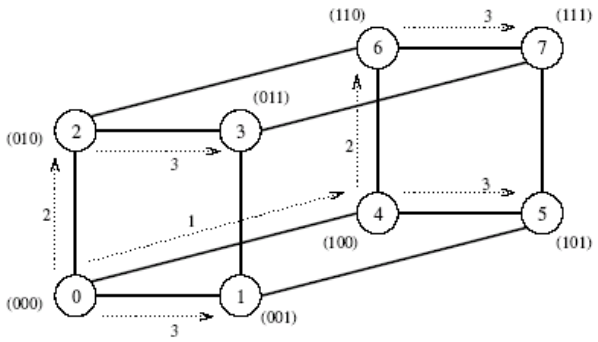
# Broadcast and Reduction Algorithms

```
1.  procedure ONE_TO_ALL_BC(d, my_id, X)
2.  begin
3.    mask := 2^d - 1;              /* Set all d bits of mask to 1 */
4.    for i := d - 1 downto 0 do    /* Outer loop */
5.      mask := mask XOR 2^i;       /* Set bit i of mask to 0 */
6.      if (my_id AND mask) = 0 then /* If lower i bits of my_id are 0 */
7.        if (my_id AND 2^i) = 0 then
8.          msg_destination := my_id XOR 2^i;
9.          send X to msg_destination;
10.       else
11.         msg_source := my_id XOR 2^i;
12.         receive X from msg_source;
13.       endelse;
14.     endif;
15.   endfor;
16. end ONE_TO_ALL_BC
```

A one-to-all broadcast procedure on a $2^d$-node network when node **0 is the source of the broadcast**. The procedure is executed at all the nodes. The procedure performs d communication steps, one along each dimension of a hypothetical hypercube. Communication proceeds from the highest to the lowest dimension . The loop counter i indicates the current dimension of the hypercube in which communication is taking place. Only the nodes with zero in the i least significant bits of their labels participate in communication along dimension i.

# Broadcast and Reduction Algorithms

```
1.  procedure ONE_TO_ALL_BC(d, my_id, X)
2.  begin
3.    mask := 2^d - 1;              /* Set all d bits of mask to 1 */
4.    for i := d - 1 downto 0 do    /* Outer loop */
5.      mask := mask XOR 2^i;       /* Set bit i of mask to 0 */
6.      if (my_id AND mask) = 0 then /* If lower i bits of my_id are 0 */
7.        if (my_id AND 2^i) = 0 then
8.          msg_destination := my_id XOR 2^i;
9.          send X to msg_destination;
10.       else
11.         msg_source := my_id XOR 2^i;
12.         receive X from msg_source;
13.       endelse;
14.     endif;
15.   endfor;
16. end ONE_TO_ALL_BC
```




Only the nodes with zero in the i least significant bits of their labels participate in communication along dimension i. For instance, on the three-dimensional hypercube shown in here, i is equal to 2 in the first time step. Therefore, only nodes 0 and 4 communicate, since their two least significant bits are zero. In the next time step, when i = 1, all nodes (that is, 0, 2, 4, and 6) with zero in their least significant bits participate in communication. The procedure terminates after communication has taken place along all dimensions.

# Broadcast and Reduction Algorithms

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

```
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
```

A modified one-to-all broadcast procedure that works for any value of source between 0 and p - 1. By performing the XOR operation at Line 3, algorithm relabels the source node to 0, and relabels the other nodes relative to the source. After this relabeling, the earlier algorithm can be applied to perform the broadcast.

# Broadcast and Reduction Algorithms

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

```
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
```

# 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:

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

innovate achieve lead

BITS Pilani
Pilani Campus

# All-to-All Broadcast and Reduction

# All-to-All Broadcast and Reduction

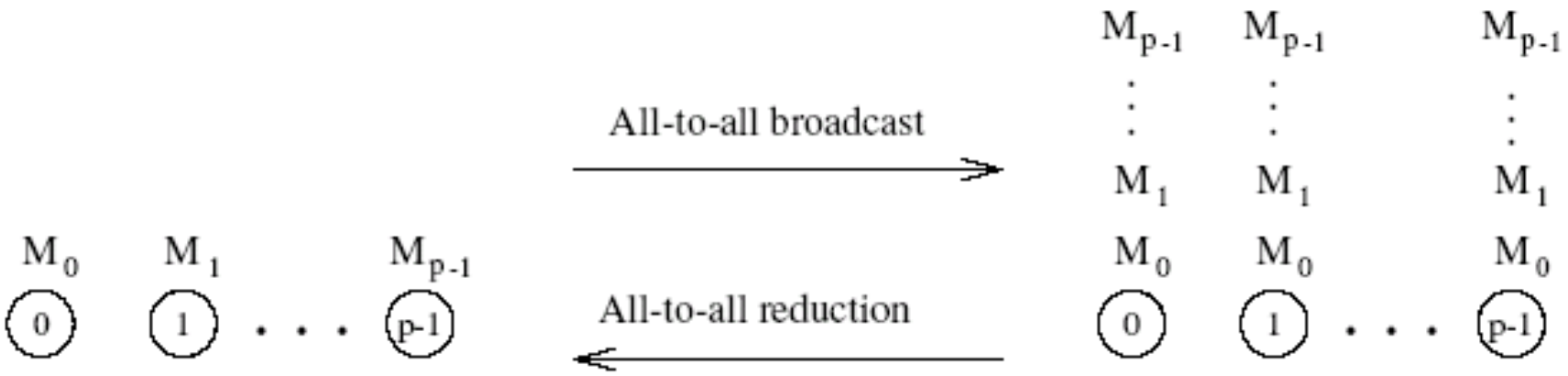
- All-to-all broadcast is a generalization of one-to-all broadcast in which all p nodes simultaneously initiate a broadcast
- A process sends the same m-word message to every other process, but different processes may broadcast different messages
- All-to-all broadcast is used in matrix operations, including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

# All-to-All Broadcast and Reduction

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

# All-to-All Broadcast and Reduction

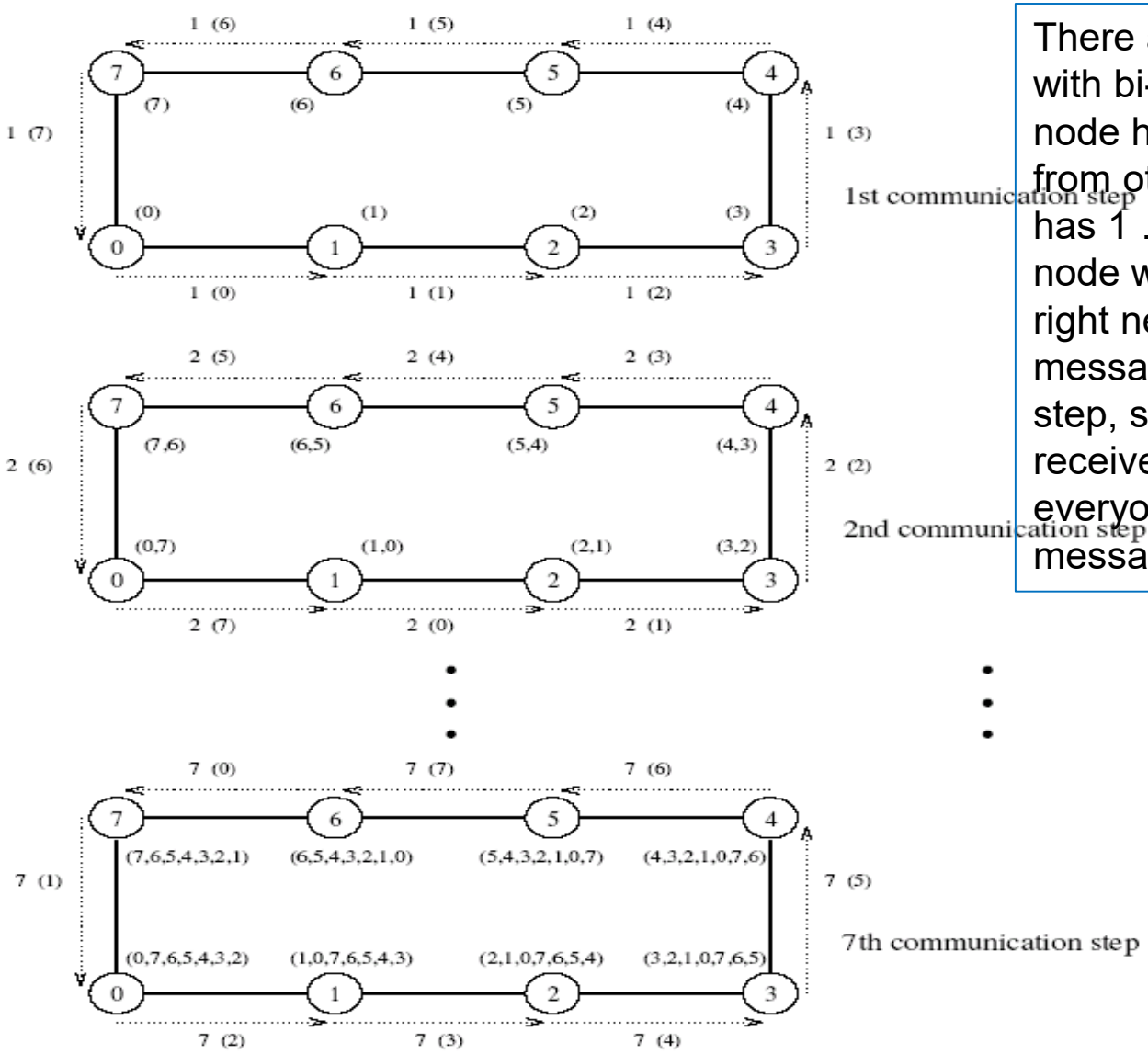
- All-to-all broadcast and all-to-all reduction.



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

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

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




There are 8 nodes in ring or array with bi-directional links. Each node has a message different from others. Node0 has 0, node1 has 1 …. In the first step, each node will send its message to the right neighbor, and receive message from left. In the second step, send the latest message received to the right. In p-1 steps, everyone will get everyone's message.

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

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

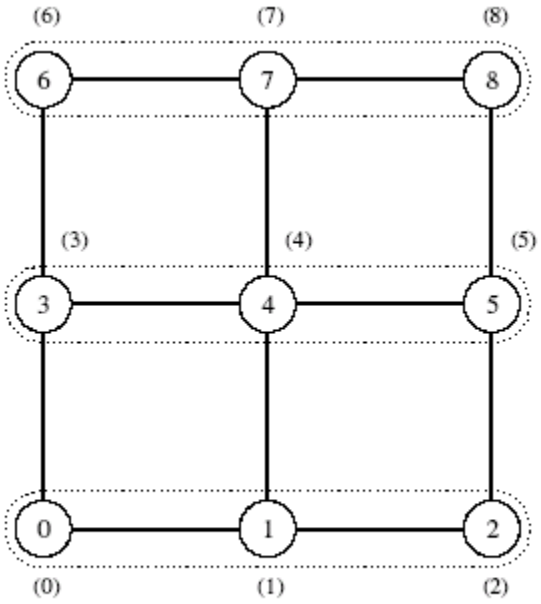
```
1.    procedure ALL_TO_ALL_BC_RING(my_id, my_msg, p, result)
2.    begin
3.        left := (my_id − 1) mod p;
4.        right := (my_id + 1) mod p;
5.        result := my_msg;
6.        msg := result;
7.        for i := 1 to p − 1 do
8.            send msg to right;
9.            receive msg from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

# All-to-all Broadcast on a Mesh

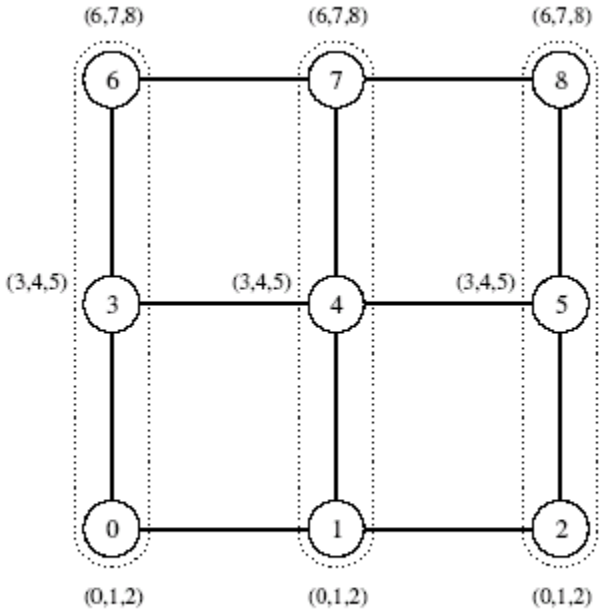
- Performed in two phases - in the first phase, each row of the mesh performs an all-to-all broadcast using the procedure for the linear array.
  - In this phase, all nodes collect $\sqrt{p}$ messages corresponding to the $\sqrt{p}$ nodes of their respective rows. Each node consolidates this information into a single message of size $m\sqrt{p}$.
- The second communication phase is a columnwise all-to-all broadcast of the consolidated messages.
  - By the end of this phase, each node obtains all p pieces of m-word data that originally resided on different nodes

# All-to-all Broadcast on a Mesh

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).




(a) Initial data distribution




(b) Data distribution after rowwise broadcast

# All-to-all Broadcast on a Mesh

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

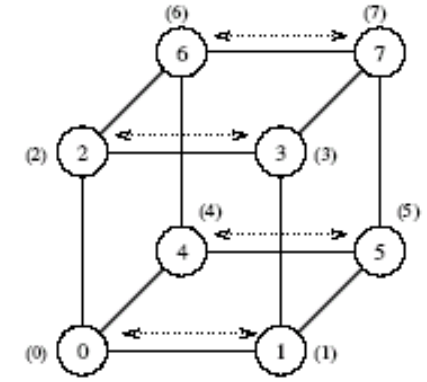
```
1.      procedure ALL_TO_ALL_BC_MESH(my_id, my_msg, p, result)
2.      begin
/* Communication along rows */
3.          left := my_id − (my_id mod √p) + (my_id − 1) mod √p;
4.          right := my_id − (my_id mod √p) + (my_id + 1) mod √p;
5.          result := my_msg;
6.          msg := result;
7.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

# All-to-all broadcast on a Hypercube

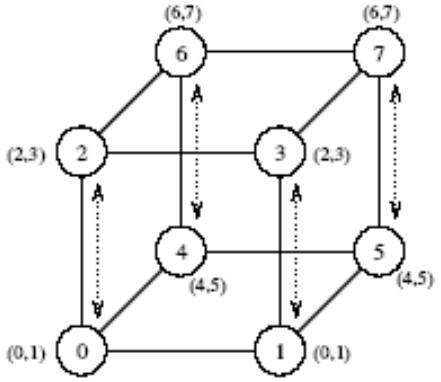
- The hypercube algorithm for all-to-all broadcast is an extension of the mesh algorithm to log p dimensions.
- The procedure requires log p steps. Communication takes place along a different dimension of the p-node hypercube in each step.
- In every step, pairs of nodes exchange their data and double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

# All-to-all broadcast on a Hypercube

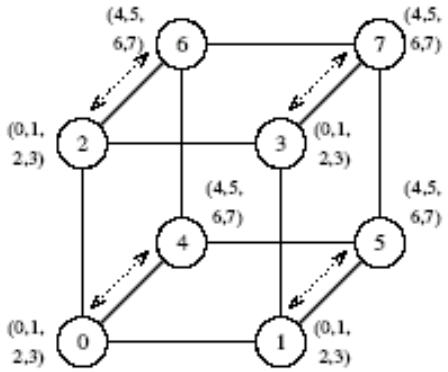
All-to-all broadcast on an eight-node hypercube.



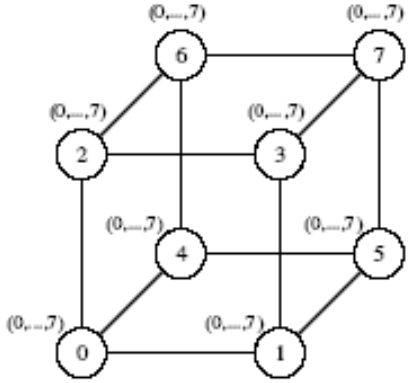

(a) Initial distribution of messages




(b) Distribution before the second step




(c) Distribution before the third step




(d) Final distribution of messages

# All-to-all broadcast on a Hypercube

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

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

Communication starts from the lowest dimension of the hypercube and then proceeds along successively higher dimensions (Line 4). In each iteration, nodes communicate in pairs so that the labels of the nodes communicating with each other in the i th iteration differ in the i th least significant bit of their binary representations (Line 5). After an iteration's communication steps, each node concatenates the data it receives during that iteration with its resident data (Line 8). This concatenated message is transmitted in the following iteration.

# 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 or a linear array, all-to-all broadcast involves p - 1 steps of communication between nearest neighbors. Each step, involving a message of size m, takes time $t_s + t_w m$.
- Therefore, the time taken by the entire operation is
  - $(t_s + t_w m)(p-1)$.
- On a mesh,
  - first phase involves $\sqrt{p}$ nodes doing all-to-all broadcast

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

  - Second phase involves $\sqrt{p}$ nodes doing all-to-all broadcast but with message size $m\sqrt{p}$

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

- On a mesh, the time is given by: $2t_s(\sqrt{p} - 1) + t_w m(p-1)$.

# Cost Analysis

- On a hypercube
  - On a p-node hypercube, the size of each message exchanged in the ith of the log p steps is $2^{i-1}m$.

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

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

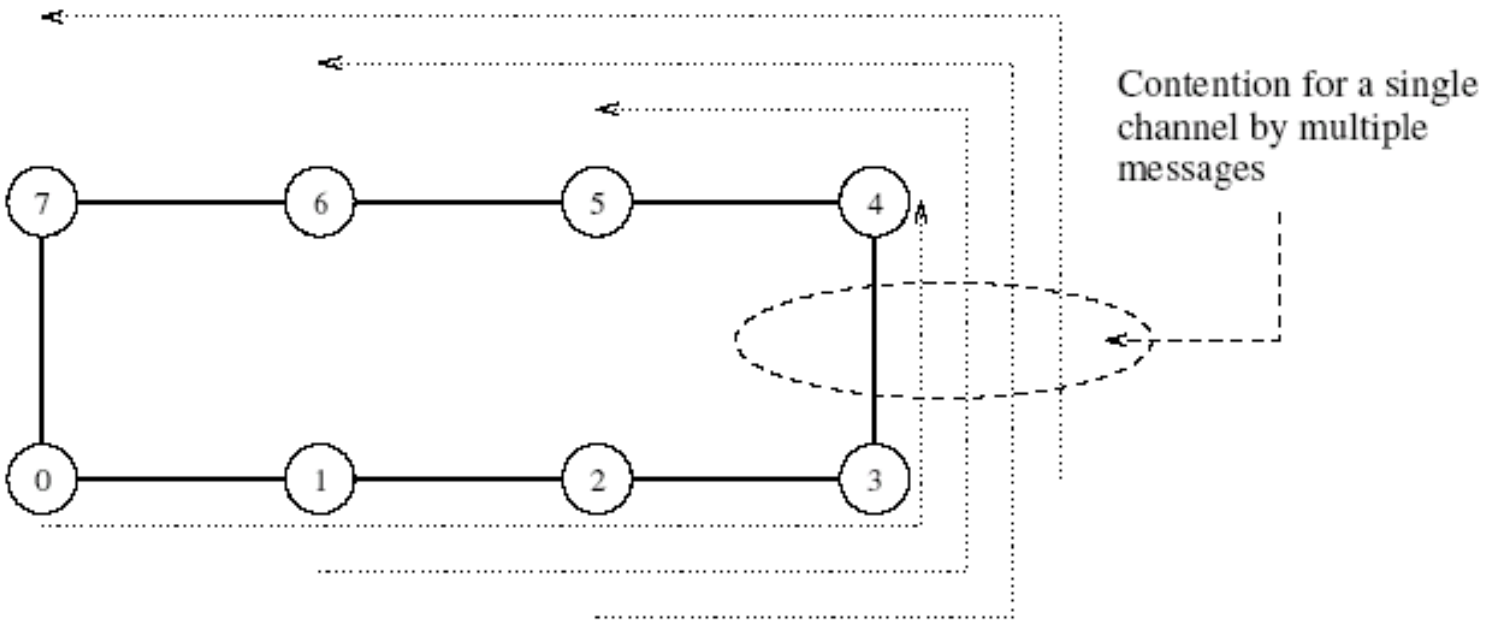
  - Term associated with $t_w$ in the expressions for the communication time of all-to-all broadcast is $t_w m(p - 1)$ for all the architectures.
  - This term also serves as a lower bound for the communication time of all-to-all broadcast for parallel computers on which a node can communicate on only one of its ports at a time. This is because each node receives at least m(p - 1) words of data, regardless of the architecture.
    - Thus, for large messages, a highly connected network like a hypercube is no better than a simple ring in performing all-to-all broadcast or all-to-all reduction.

# All-to-all broadcast: Notes

- Another noteworthy property of all-to-all broadcast is that, unlike one-to-all broadcast, the hypercube algorithm cannot be applied unaltered to mesh and ring architectures. The reason is that the hypercube procedure for all-to-all broadcast would cause congestion on the communication channels of a smaller-dimensional network with the same number of nodes.

# All-to-all broadcast: Notes

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



innovate achieve lead

BITS Pilani
Pilani Campus

# All-Reduce and Prefix-Sum Operations

# All-Reduce and Prefix-Sum Operations

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

# All-Reduce

- All-reduce
  - A simple method to perform all-reduce is to perform an all-to-one reduction followed by a one-to-all broadcast
    - However, there is a faster way to perform all-reduce by using the communication pattern of all-to-all broadcast.
  - Using the pattern of all-to-all broadcast
    - Unlike all-to-all broadcast, each message transferred in the reduction operation has only one word
    - The size of the messages does not double in each step because the numbers are added instead of being concatenated
    - Time for this operation on hypercube is $(t_s + t_w m) \log p$.

All to all broadcast on Hypercube

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

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

All-reduce on Hypercube

$$T = \sum_{i=1}^{logp} t_s + twm$$

$$= tslogp + twmlogp$$

$$= (t_s + twm)logp$$

# All-Reduce

- All-reduce
  - A simple method to perform all-reduce is to perform an all-to-one reduction followed by a one-to-all broadcast
    - However, there is a faster way to perform all-reduce by using the communication pattern of all-to-all broadcast.
  - Using the pattern of all-to-all broadcast
    - Unlike all-to-all broadcast, each message transferred in the reduction operation has only one word
    - The size of the messages does not double in each step because the numbers are added instead of being concatenated

All to all broadcast on Mesh

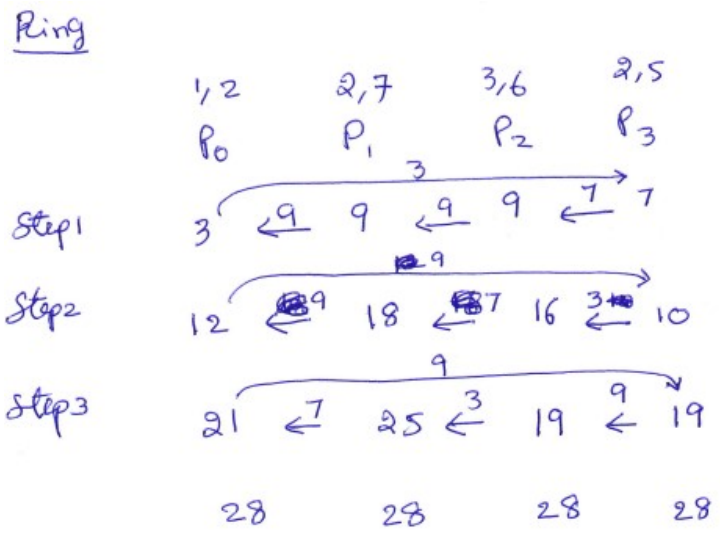
Step1:$(t_s + twm)(\sqrt{p} - 1)$
Step2:$\left(t_s + twm\sqrt{p}\right)\left(\sqrt{p} - 1\right)$
$T = 2ts(\sqrt{p} - 1) + twm(p - 1)$

All-reduce on Mesh

Step1:$(t_s + twm)(\sqrt{p} - 1)$
Step2:$(t_s + twm)\left(\sqrt{p} - 1\right)$
$T = 2(\sqrt{p} - 1)(ts + t_w m)$

# All-Reduce

- All-reduce
  - Using the pattern of all-to-all broadcast
    - Unlike all-to-all broadcast, each message transferred in the reduction operation has only one word
    - The size of the messages does not double in each step because the numbers are added instead of being concatenated




All to all broadcast on Ring

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

All-reduce on Ring

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

# The Prefix-Sum Operation(also known as the scan operation)

- Another important problem that can be solved by using a communication pattern similar to that used in all-to-all broadcast and all-reduce operations
- Given $p$ numbers $n_0,n_1,\ldots,n_{p-1}$ (one on each node), the problem is to compute the sums $s_k = \sum_{i=0}^{k} n_i$ 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 $S_k$.
  - For example, if the original sequence of numbers is <3, 1, 4, 0, 2>, then the sequence of prefix sums is <3, 4, 8, 8, 10>
- The operation can be implemented using the all-to-all broadcast.
  - 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 $k$.

# The Prefix-Sum Operation(also known as the scan operation)

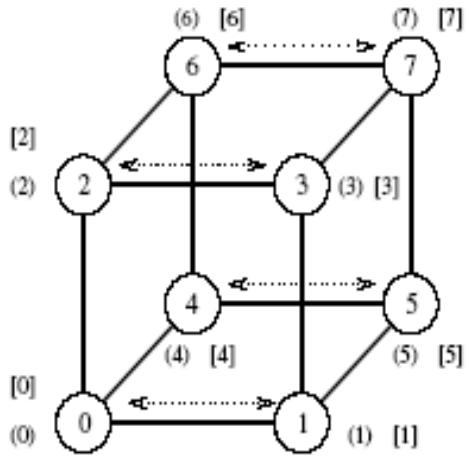
- Ring
  - Similar to All-reduce except that add only if label<my_id
  - Cost: T=(ts + twm)(p-1)
- Mesh
  - Similar to All-reduce except that add only if label<my_id

Step1:$(t_s + twm)(\sqrt{p} - 1)$
Step2:$(t_s + twm)\left(\sqrt{p} - 1\right)$
$T = 2(\sqrt{p} - 1)(ts + t_w m)$

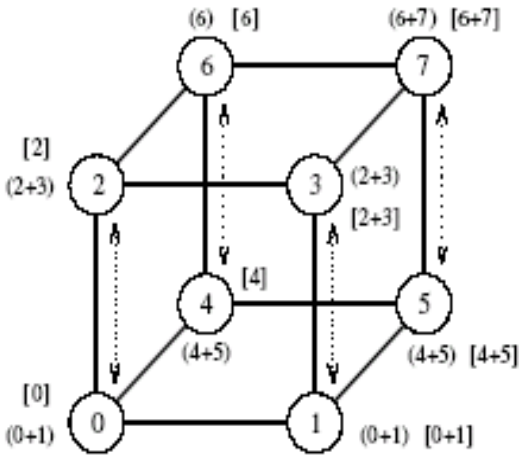
- Hypercube
  - Same as All-reduce except that sum up only if label<my_id

$$T = \sum_{i=1}^{logp} t_s + twm$$
$$= tslogp + twmlogp$$
$$= (t_s + twm)logp$$

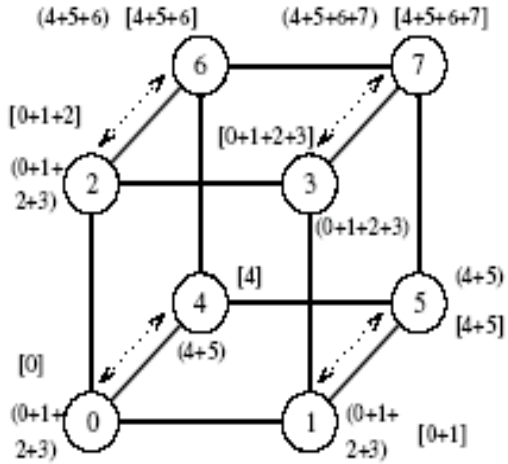
# The Prefix-Sum Operation



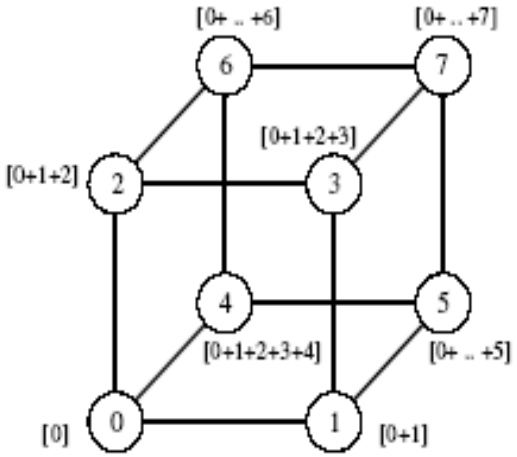

(a) Initial distribution of values




(b) Distribution of sums before second ste



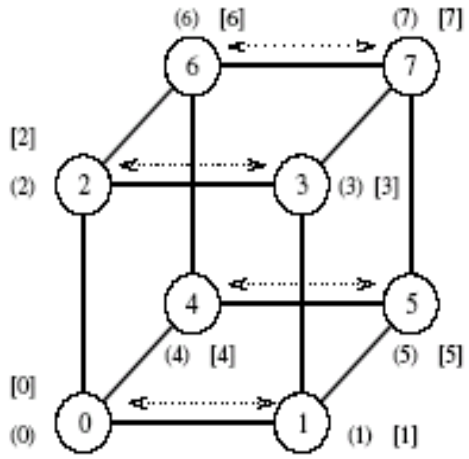

(c) Distribution of sums before third step



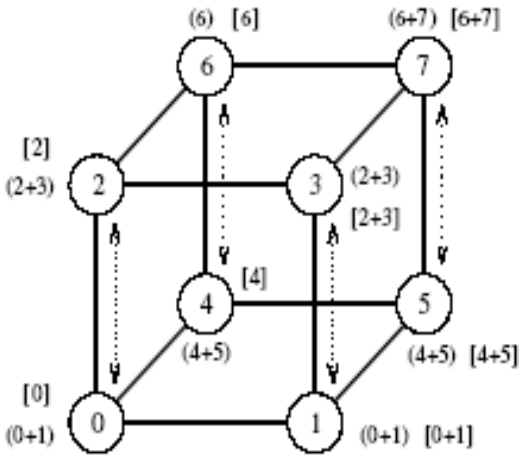

(d) Final distribution of prefix sums

Node with label k uses information from only the k-node subset of those nodes whose labels are less than or equal to k. To accumulate the correct prefix sum, every node maintains an additional result buffer. This buffer is denoted by square brackets in figure. At the end of a communication step, the content of an incoming message is added to the result buffer only if the message comes from a node with a smaller label than that of the recipient node. The contents of the outgoing message (denoted by parentheses in the figure) are updated with every incoming message, just as in the case of the all-reduce operation

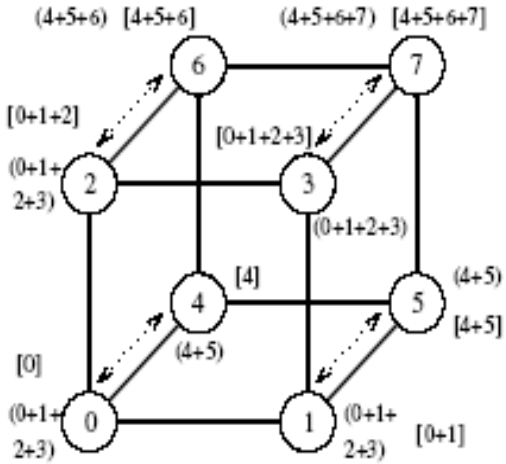
# The Prefix-Sum Operation



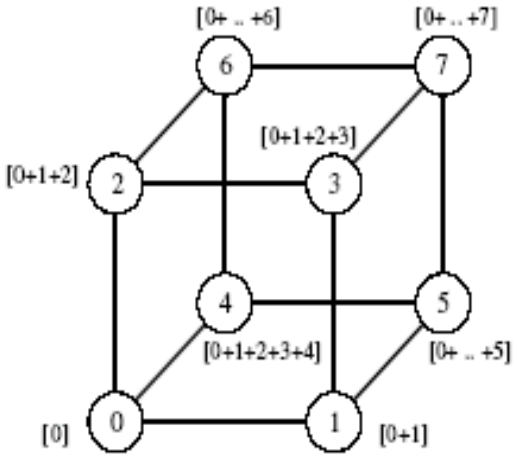

(a) Initial distribution of values




(b) Distribution of sums before second ste




(c) Distribution of sums before third step




(d) Final distribution of prefix sums

After the first communication step, nodes 0, 2, and 4 do not add the data received from nodes 1, 3, and 5 to their result buffers. However, the contents of the outgoing messages for the next step are updated

# The Prefix-Sum Operation

- Since not all of the messages received by a node contribute to its final result, some of the messages it receives may be redundant
  - the presence or absence of these messages does not affect the results of the algorithm

# The Prefix-Sum Operation

Prefix sums on a *d*-dimensional hypercube.

```
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
```

innovate achieve lead

BITS Pilani
Pilani Campus

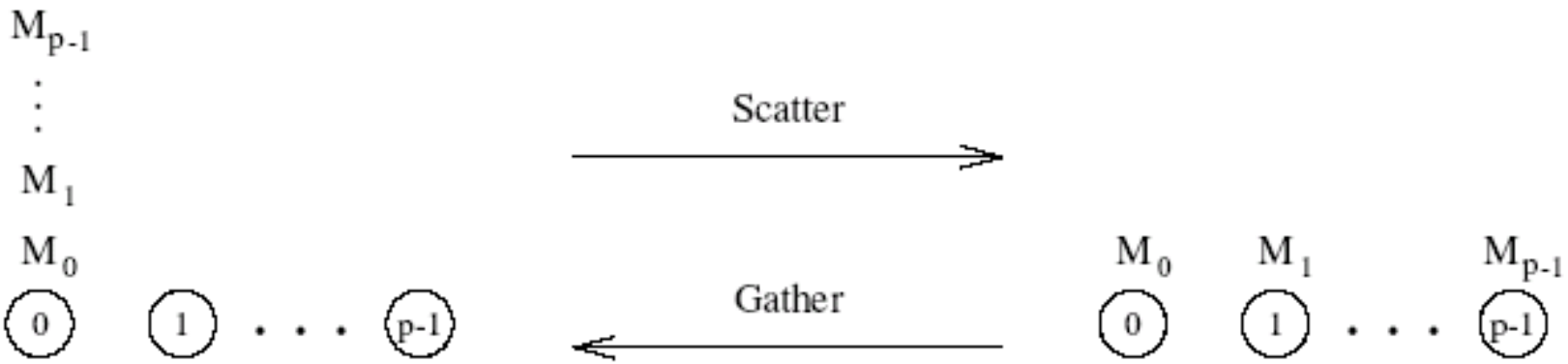
# Scatter (one-to-all personalized communication) and Gather

# 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
- While the scatter operation is fundamentally different from broadcast, the algorithmic structure is similar, except for differences in message sizes
  - Unlike one-to-all broadcast, one-to-all personalized communication does not involve any duplication of data
  - messages get smaller in scatter and stay constant in broadcast
- In the *gather* operation, a single node collects a unique message from each node.
  - The gather operation is exactly the inverse of the scatter operation and can be executed as such.

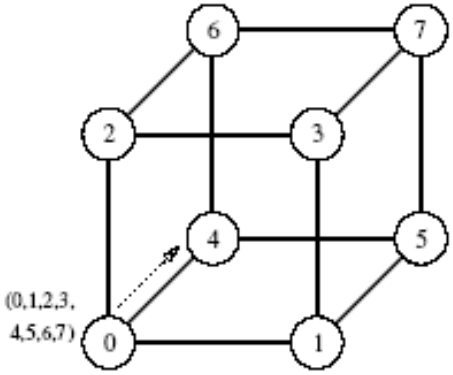
# Gather and Scatter Operations

Scatter and gather operations.

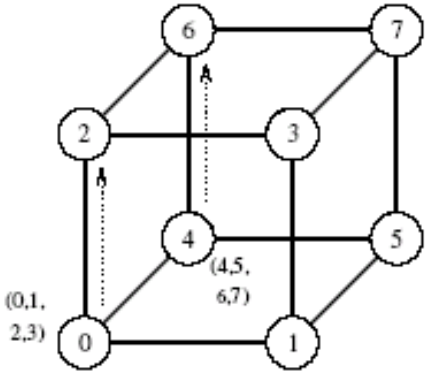


# Example of the Scatter Operation

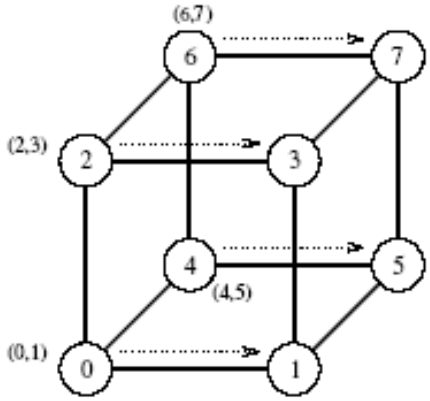
The scatter operation on an eight-node hypercube.



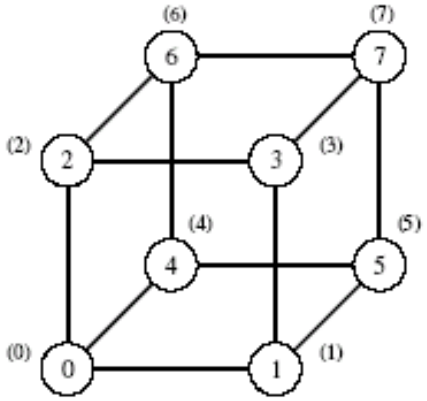

(a) Initial distribution of messages




(b) Distribution before the second step




(c) Distribution before the third step




(d) Final distribution of messages

The source node (node 0) contains all the messages. The messages are identified by the labels of their destination nodes. In the first communication step, the source transfers half of the messages to one of its neighbors. In subsequent steps, each node that has some data transfers half of it to a neighbor that has yet to receive any data. There is a total of log p communication steps corresponding to the log p dimensions of the hypercube.

# Example of the Scatter Operation

The scatter operation on an eight-node hypercube.

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

# Cost of Scatter and Gather

- Hypercube
  - All links of a p-node hypercube along a certain dimension join two p/2-node subcubes.
  - In each communication step of the scatter operations, data flow from one subcube to another. The data that a node owns before starting communication in a certain dimension are such that half of them need to be sent to a node in the other subcube.
  - In every step, a communicating node keeps half of its data, meant for the nodes in its subcube, and sends the other half to its neighbor in the other subcube. The time in which all data are distributed to their respective destinations is
  - $T = ts\ log\ p + twm\ (p-1)$
    - This is similar to calculation of all-to-all boradcats

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

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

- This time holds for a linear array as well as a 2-D mesh.
  - In linear array, just like broadcast, but message size reduces by half everytime.

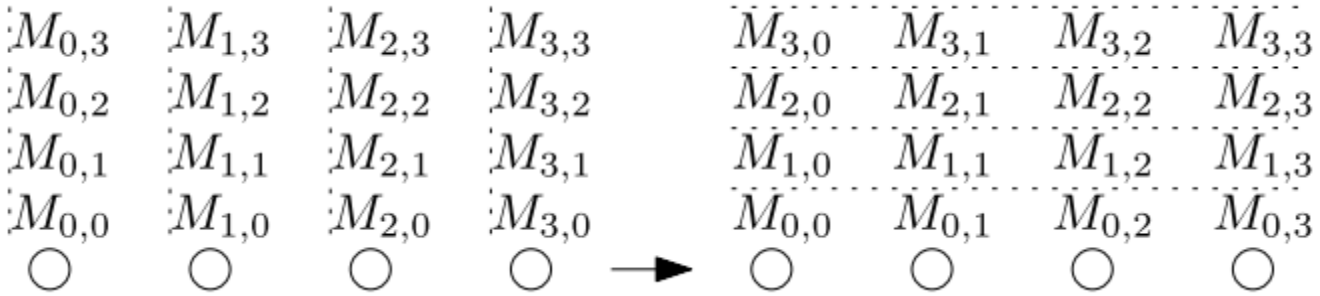
innovate achieve lead

BITS Pilani
Pilani Campus

# All-to-All Personalized Communication

# All-to-All Personalized Communication

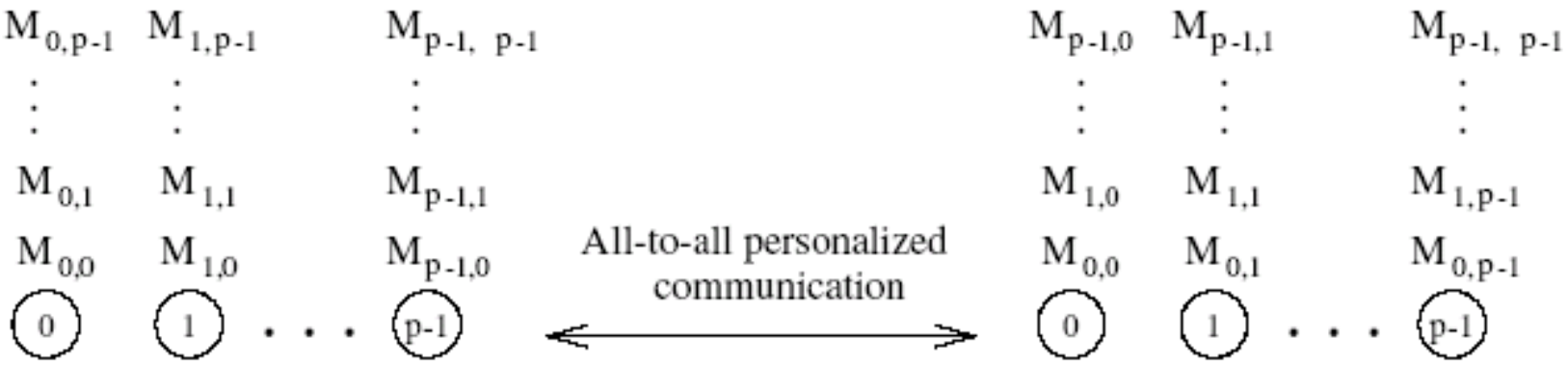
- Each node has a distinct message of size *m* for every other node.
  - This is unlike all-to-all broadcast, in which each node sends the same message to all other nodes.
- All-to-all personalized communication is also known as *total exchange*.




  - A careful observation of this figure would reveal that this operation is equivalent to transposing a two-dimensional array of data distributed among p processes using 1-dmesional array partioning

# All-to-All Personalized Communication

All-to-all personalized communication.

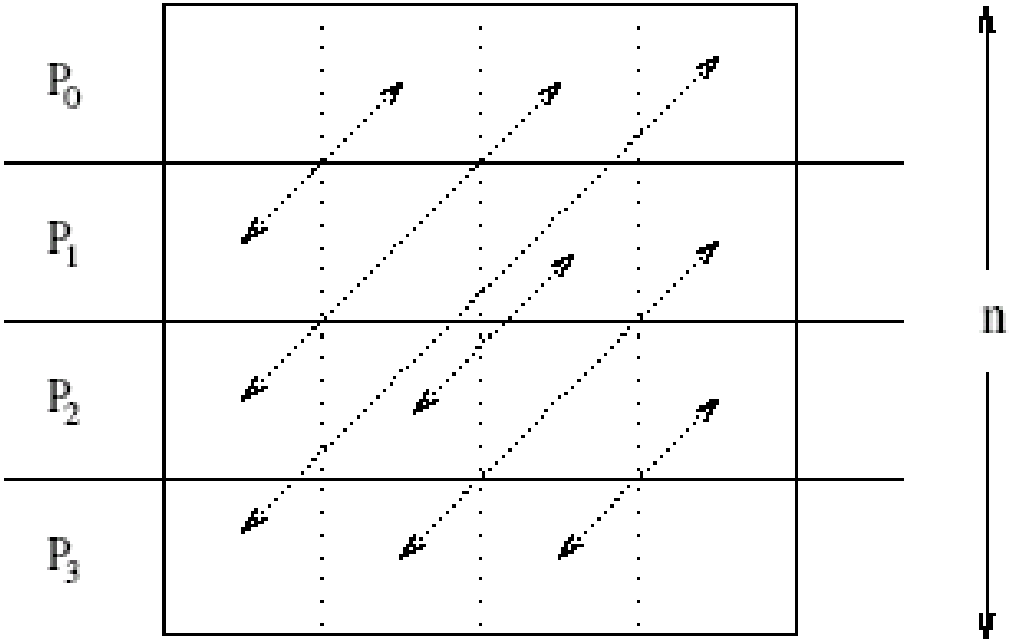


# All-to-All Personalized Communication: Example

- Consider the problem of transposing a matrix.
- Consider an n x n matrix mapped onto n processors such that each processor contains one full row of the matrix.
  - With this mapping, processor $P_i$ initially contains the elements of the matrix with indices [i, 0], [i, 1], ..., [i, n - 1].
  - After the transposition, element [i, 0] belongs to $P_0$, element [i, 1] belongs to $P_1$, and so on.
  - In general, element [i, j] initially resides on $P_i$ , but moves to $P_j$ during the transposition.
- The 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 x 4* matrix using four processes.



# All-to-All Personalized Communication on a Ring

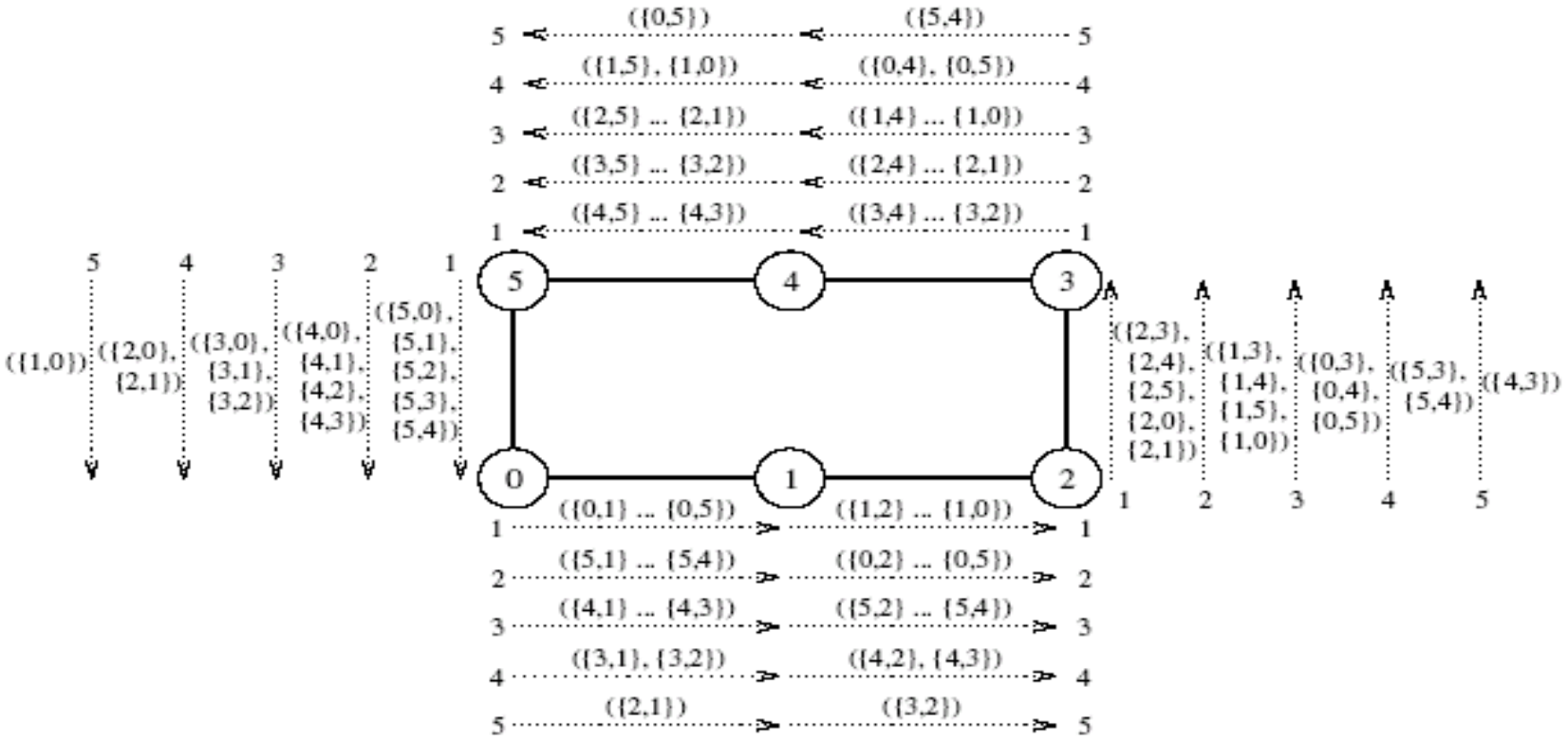
- The communication patterns of all-to-all personalized communication are identical to those of all-to-all broadcast on all three architectures.
  - Only the size and the contents of messages are different

# 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 *({$x_1$,$y_1$}, {$x_2$,$y_2$},…, {$x_n$,$y_n$}*, 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:

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

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

- In the all-to-all personalized communication procedure, each node sends m(p - 1) words of data because it has an m-word packet for every other node
  - Assume that all messages are sent either clockwise or counterclockwise
  - The average distance that an m-word packet travels is $\frac{\sum_{i=1}^{p-1} i}{p-1} = \frac{p}{2}$.
  - If each node sends $p - 1$ m-words,
    - Total number of m-words$= p * (p - 1)$
    - If each m-word travels p/2 links on an average, then total traffic $p * (p - 1) * \frac{p}{2} * m$
    - Number of links in the network$= p$
    - Optimal Communication time$= \frac{t_w * p * (p-1) * \frac{p}{2} * m}{\mathrm{p}} = \frac{\mathrm{t_w m(p-1)p}}{2}$

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

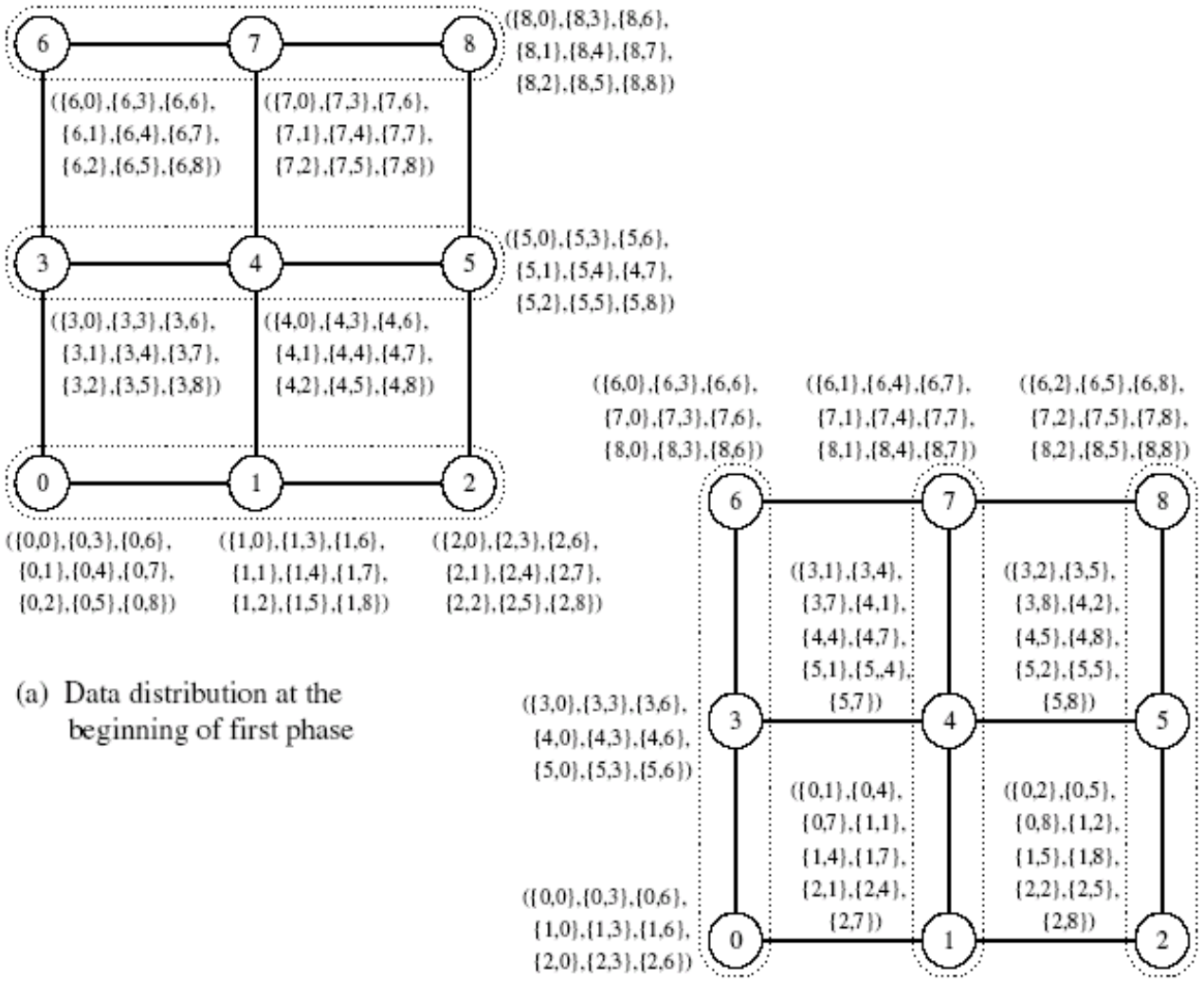
- In the all-to-all personalized communication procedure, each node sends m(p - 1) words of data because it has an m-word packet for every other node
  - Optimal Communication time= $\frac{t_w * p * (p-1) * \frac{p}{2} * m}{p} = \frac{t_w mp(p-1)}{2}$
  - Neglecting $t_s$, time taken by algorithm on ring is same as optimal time.

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

# All-to-All Personalized Communication on a Mesh

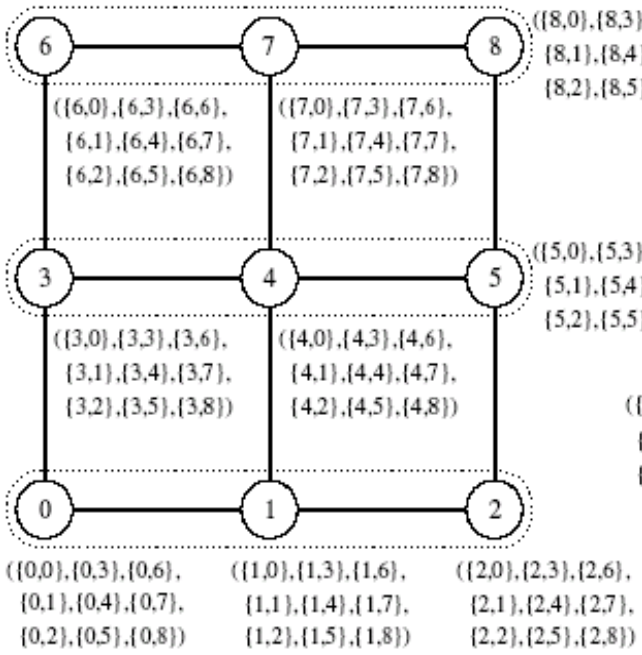
- Each node first groups its $p$ messages according to the columns of their destination nodes.
- All-to-all personalized communication is performed independently in each row with clustered messages of size $m\sqrt{p}$.
- Messages in each node are sorted again, this time according to the rows of their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

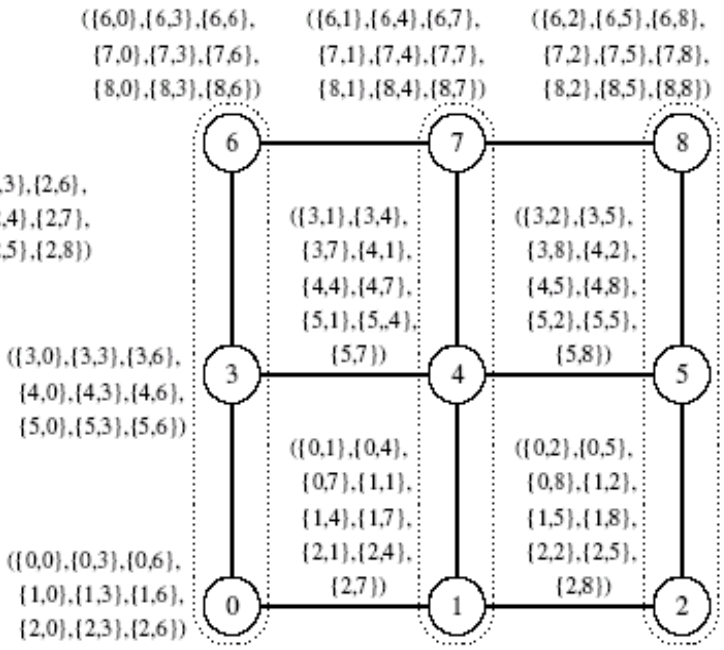



a 3 x 3 mesh, in which every node initially has nine m-word messages, one meant for each node. Each node assembles its data into three groups of three messages each: The first group contains the messages destined for nodes labeled 0, 3, and 6; the second group contains the messages for nodes labeled 1, 4, and 7; and the last group has messages for nodes labeled 2, 5, and 8.

# All-to-All Personalized Communication on a Mesh




(a) Data distribution at the beginning of first phase




(b) Data distribution at the beginning of second phase

After the messages are grouped, all-to-all personalized communication is performed independently in each row with clustered messages of size $m\sqrt{p}$

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

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

- In the all-to-all personalized communication procedure, each node sends m(p - 1) words of data because it has an m-word packet for every other node
  - The average distance that an m-word packet travels is (row-wise + column-wise)=$\frac{\sqrt{p}(\sqrt{p}-1)\left(\frac{\sqrt{p}}{2}\right)+\sqrt{p}(\sqrt{p}-1)\left(\frac{\sqrt{p}}{2}\right)}{p-1} = \frac{(p-\sqrt{p})(\sqrt{p})}{(p-1)}$.
  - If each node sends $p-1$ m-words,
    - Total number of m-words$= p*(p-1)$
    - Total traffic $p*(p-1)*\frac{(p-\sqrt{p})(\sqrt{p})}{(p-1)}*m$
    - Number of links in the network$= 2p$
    - Optimal Communication time= $\frac{p*(p-1)*\frac{(p-\sqrt{p})(\sqrt{p})}{(p-1)}*m}{2\mathrm{p}} = \frac{(p\sqrt{p}-p)\mathrm{m}}{2} = \frac{\mathrm{mp}(\sqrt{p}-1)}{2}$

# 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.,

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

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

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

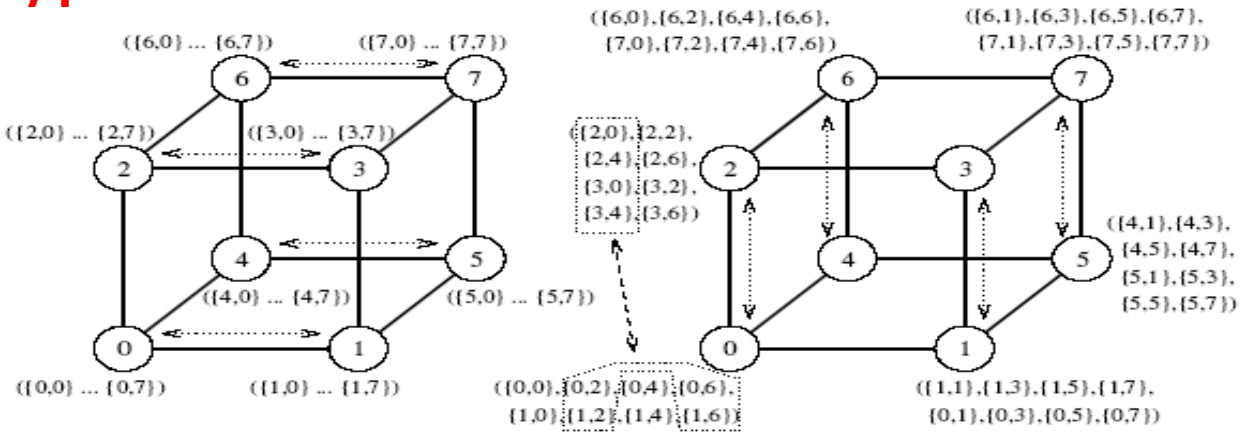
- In the all-to-all personalized communication procedure, each node sends m(p - 1) words of data because it has an m-word packet for every other node
  - Optimal Communication time= $\frac{\mathrm{mp}(\sqrt{p}-1)}{2}$
  - Taking $t_s$=0, time taken by algorithm on Mesh is optimal within a small (<=4)constant factor

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

# All-to-All Personalized Communication on a Hypercube

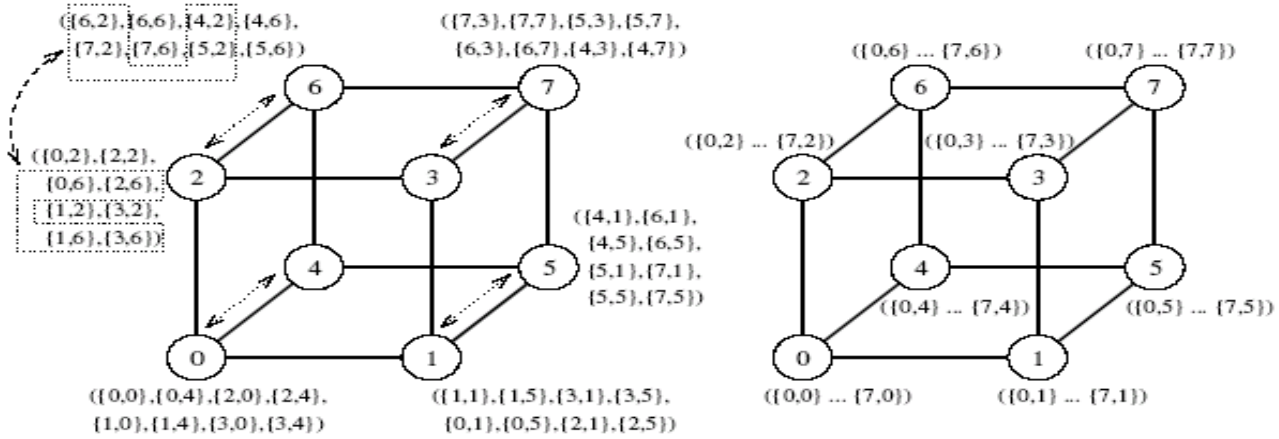
- Generalize the mesh algorithm to *log p* steps.
- At any stage in all-to-all personalized communication, every node holds *p* packets of size *m* each.
- While communicating in a particular dimension, every node sends *p/2* of these packets (consolidated as one message).
- A node must rearrange its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

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

- In the all-to-all personalized communication procedure, each node sends m(p - 1) words of data because it has an m-word packet for every other node
  - The average distance that an m-word packet travels is (log p)/2.
  - If each node sends $p - 1$ m-words,
    - Total number of m-words= $p * (p - 1)$
    - Total traffic $p * (p - 1) * (logp)/2 * m$
    - Number of links in the network= $(plogp)/2$
    - Optimal Communication time= $\frac{\mathrm{p(p-1)(logp/2)*m}}{(plogp)/2} = (\mathrm{p-1})\mathrm{m}$

# 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:

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

- Optimal cost is $\frac{\mathrm{p(p-1)(logp/2)*m}}{(plogp)/2} = \mathrm{m(p-1)}$
- This is not optimal!

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

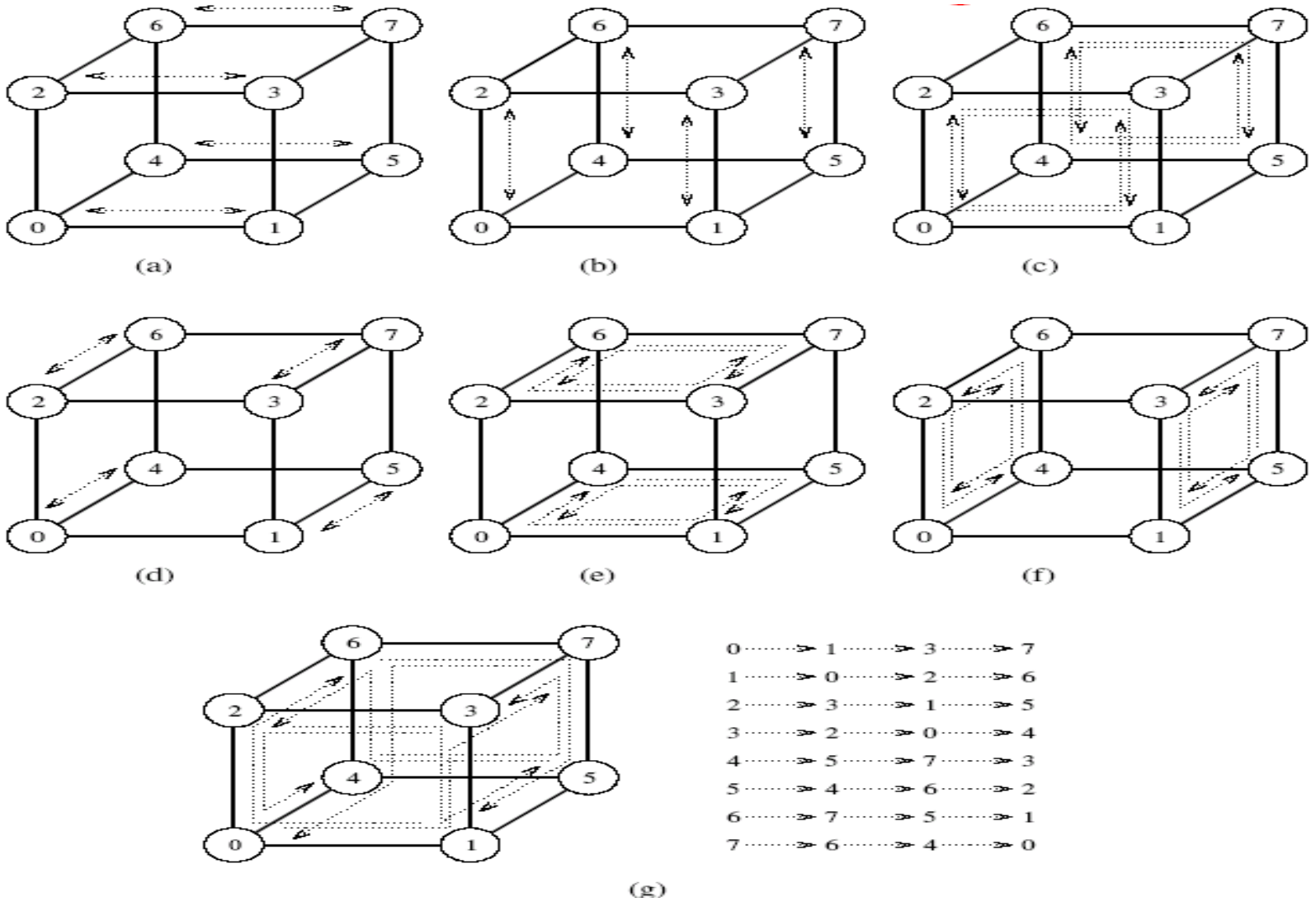
- Each node simply performs $p - 1$ communication steps, exchanging $m$ words of data with a different node in every step.
- A node must choose its communication partner in each step so that the hypercube links do not suffer congestion.
- In the $j^{th}$ communication step, node $i$ exchanges data with 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.

# E-cube routing

- A message traveling from node i to node j on a hypercube must pass through at least l links, where l is the Hamming distance between i and j
  - that is, the number of nonzero bits in the binary representation of (i XOR j)
  - A message traveling from node i to node j traverses links in l dimensions (corresponding to the nonzero bits in the binary representation of (i XOR j)).
  - Although the message can follow one of the several paths of length l that exist between i and j (assuming l > 1), a distinct path is obtained by sorting the dimensions along which the message travels in ascending order. According to this strategy, the first link is chosen in the dimension corresponding to the least significant nonzero bit of (i XOR j), and so on
  - This routing scheme is known as E-cube routing.

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

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



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

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

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

| i | | XOR 0 | Node0 talks to | XOR 1 | Node1 talks to | XOR 2 | Node2 talks to | XOR 3 | Node3 talks to | XOR 4 | Node4 talks to | XOR 5 | Node5 talks to | XOR 6 | Node6 talks to | Xor 7 | Node7 talks to |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 001 | 1 | 1 | 0 | 0 | 11 | 3 | 10 | 2 | 101 | 5 | 100 | 4 | 111 | 7 | 110 | 6 |
| 2 | 010 | 10 | 2 | 11 | 3 | 0 | 0 | 1 | 1 | 110 | 6 | 111 | 7 | 100 | 4 | 101 | 5 |
| 3 | 011 | 11 | 3 | 10 | 2 | 1 | 1 | 0 | 0 | 111 | 7 | 110 | 6 | 101 | 5 | 100 | 4 |
| 4 | 100 | 100 | 4 | 101 | 5 | 110 | 6 | 111 | 7 | 0 | 0 | 1 | 1 | 10 | 2 | 11 | 3 |
| 5 | 101 | 101 | 5 | 100 | 4 | 111 | 7 | 110 | 6 | 1 | 1 | 0 | 0 | 11 | 3 | 10 | 2 |
| 6 | 110 | 110 | 6 | 111 | 7 | 100 | 4 | 101 | 5 | 10 | 2 | 11 | 3 | 0 | 0 | 1 | 1 |
| 7 | 111 | 111 | 7 | 110 | 6 | 101 | 5 | 100 | 4 | 11 | 3 | 10 | 2 | 1 | 1 | 0 | 0 |

# 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:

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

- This is asymptotically optimal in message size.

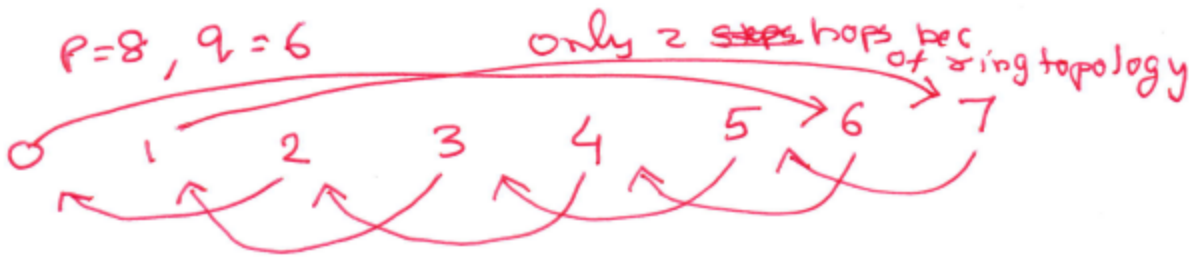

# Circular Shift

# Circular Shift

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

# Circular Shift on a Ring

- The implementation on a ring is rather intuitive
- Messages are sent to a unique node q steps away
  - Beyond q=p/2, nodes can be reached using wraparound links in smaller hops.
- It can be performed in min{*q%p, p – q%p*} neighbor communications
  - Maximum communication cost can be p/2




Ring topology p= 8

| q | q%p | p-q%p | min |
|---|---|---|---|
| 1 | 1 | 7 | 1 |
| 2 | 2 | 6 | 2 |
| 3 | 3 | 5 | 3 |
| 4 | 4 | 4 | 4 |
| 5 | 5 | 3 | 3 |
| 6 | 6 | 2 | 2 |
| 7 | 7 | 1 | 1 |
| 8 | 0 | 8 | 0 |
| 9 | 1 | 7 | 1 |
| 10 | 2 | 6 | 2 |
| 11 | 3 | 5 | 3 |
| 12 | 4 | 4 | 4 |
| 13 | 5 | 3 | 3 |
| 14 | 6 | 2 | 2 |
| 15 | 7 | 1 | 1 |
| 16 | 0 | 8 | 0 |
| 17 | 1 | 7 | 1 |

# Circular Shift on a Mesh

- If the nodes of the mesh have row-major labels, a circular q-shift can be performed on a p-node square wraparound mesh in two stages
  - First row-wise: In each row shift data by $q\%\sqrt{p}$ steps.
    - Same as circular shift ring topology
    - Maximum comm time=$\frac{\sqrt{p}}{2}$
  - Second step: During the circular row shifts, some of the data traverse the wraparound connection from the highest to the lowest labeled nodes of the rows. All such data packets must shift an additional step forward along the columns to compensate for.
    - The highest label in each row should move to the first column in the next row as per row-major order
    - Communication cost is 1.
  - Third step: In each column, shift by $\lfloor q/\sqrt{p} \rfloor$ steps
    - Same as circular shift ring topology
    - Maximum comm time=$\frac{\sqrt{p}}{2}$. This is irrespective of number of steps.

# Circular Shift on a Mesh

- If the nodes of the mesh have row-major labels, a circular q-shift can be performed on a p-node square wraparound mesh in two stages
- Maximum time

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

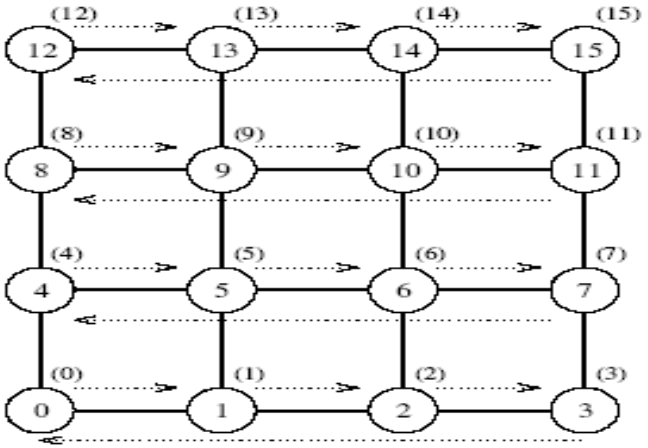
Row-wise shift: p=16 16 Column-wise

| q | q%sqrt(p) | sqrt(p)-(q%sqrt(p) | Min |
|---|---|---|---|
| 1 | 1 | 3 | 1 |
| 2 | 2 | 2 | 2 |
| 3 | 3 | 1 | 1 |
| 4 | 0 | 4 | 0 |
| 5 | 1 | 3 | 1 |
| 6 | 2 | 2 | 2 |
| 7 | 3 | 1 | 1 |
| 8 | 0 | 4 | 0 |
| 9 | 1 | 3 | 1 |
| 10 | 2 | 2 | 2 |
| 11 | 3 | 1 | 1 |
| 12 | 0 | 4 | 0 |
| 13 | 1 | 3 | 1 |
| 14 | 2 | 2 | 2 |
| 15 | 3 | 1 | 1 |
| 16 | 0 | 4 | 0 |

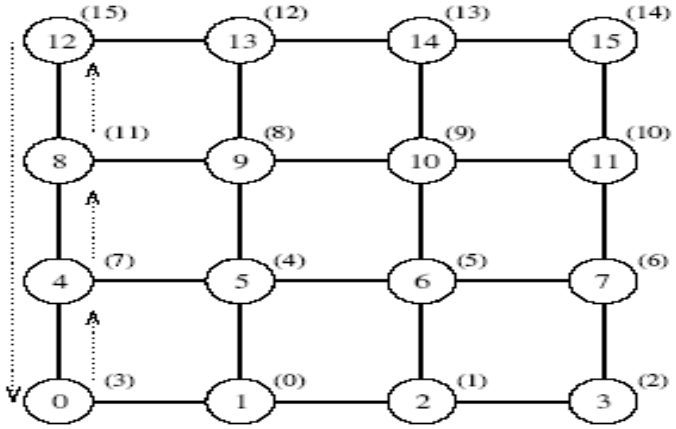
| q | FLOOR(q/sqrt(p)) | sqrt(p)-(q%sqrt(p) | Min |
|---|---|---|---|
| 1 | 0 | 4 | 0 |
| 2 | 0 | 4 | 0 |
| 3 | 0 | 4 | 0 |
| 4 | 1 | 3 | 1 |
| 5 | 1 | 3 | 1 |
| 6 | 1 | 3 | 1 |
| 7 | 1 | 3 | 1 |
| 8 | 2 | 2 | 2 |
| 9 | 2 | 2 | 2 |
| 10 | 2 | 2 | 2 |
| 11 | 2 | 2 | 2 |
| 12 | 3 | 1 | 1 |
| 13 | 3 | 1 | 1 |
| 14 | 3 | 1 | 1 |
| 15 | 3 | 1 | 1 |
| 16 | 4 | 0 | 0 |

# Circular Shift on a Mesh

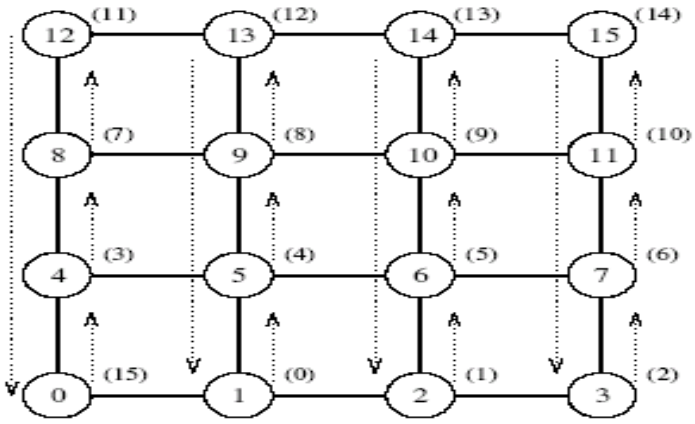
The communication steps in a circular 5-shift on a *4 x 4* mesh.



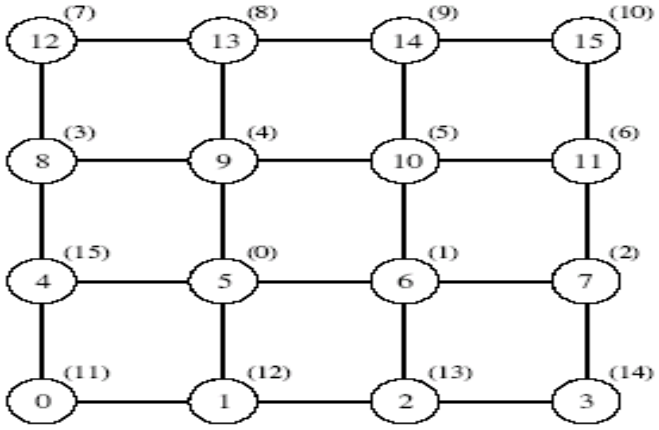

(a) Initial data distribution and the first communication step




(b) Step to compensate for backward row shifts




(c) Column shifts in the third communication step




(d) Final distribution of the data

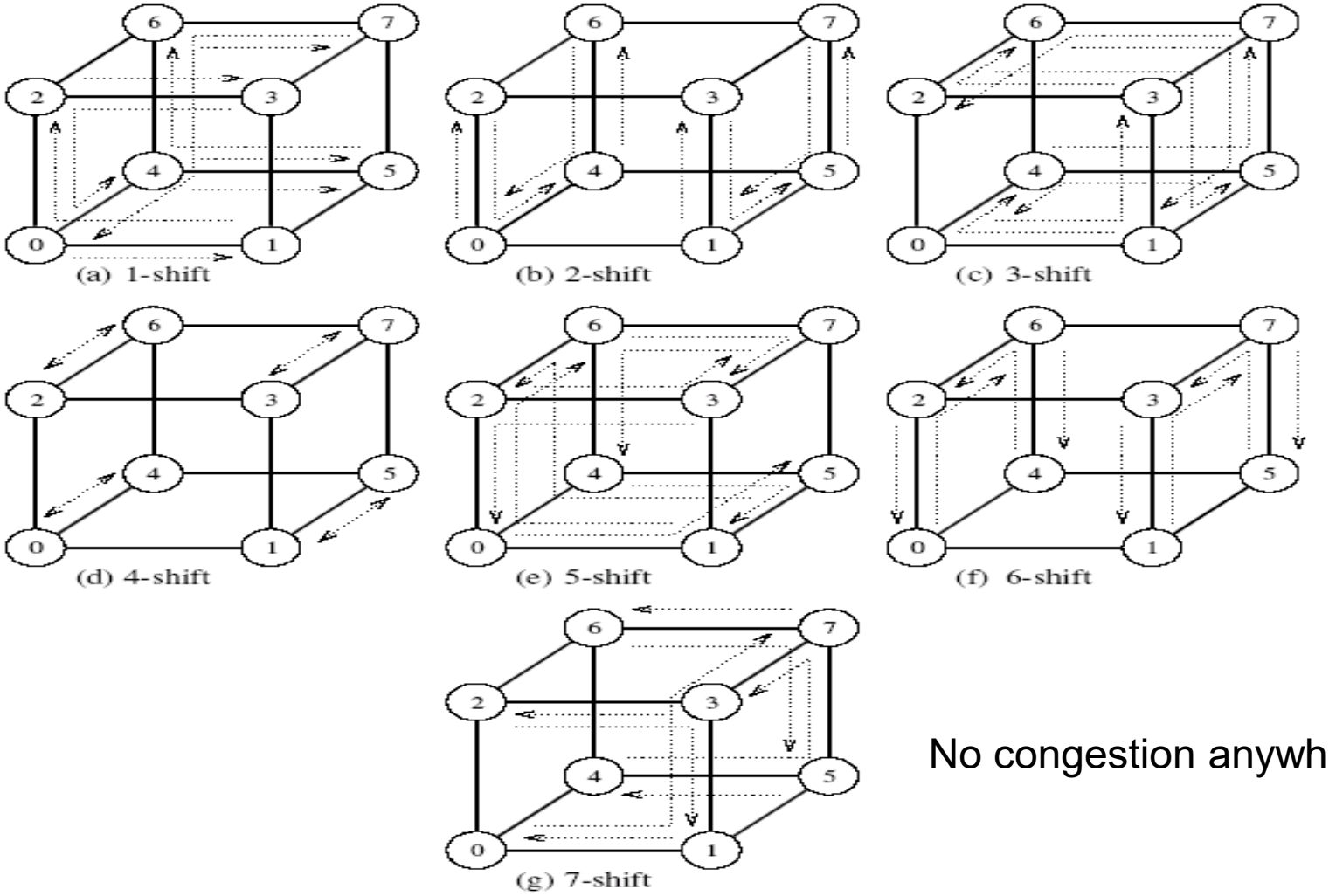
# Circular Shift on a Hypercube

- Map a linear array with $2^d$ nodes onto a $d$-dimensional hypercube.
  - We do this by assigning node i of the linear array to node j of the hypercube such that j is the d-bit binary reflected Gray code (RGC) of i.
- If $q$ is the sum of $s$ distinct powers of 2, then the circular $q$-shift on a hypercube is performed in $s$ phases
  - To perform a q-shift, we expand q as a sum of distinct powers of 2. The number of terms in the sum is the same as the number of ones in the binary representation of q.
    - For example, the number 5 can be expressed as $2^2 + 2^0$. These two terms correspond to bit positions 0 and 2 in the binary representation of 5, which is 101.
  - 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

$$T = t_s + t_w m.$$

# Circular Shift on a Hypercube using E-cube routing

Circular $q$-shifts on an $8$-node hypercube for $1 \le q < 8$.




No congestion anywhere

innovate achieve lead

BITS Pilani
Pilani Campus

# Summary

# Communication Operations and Costs

BITS Pilani

| Algo | Ring | 2D Mesh | Hypercube |
|---|---|---|---|
| One-to-all broadcast | $(t_s+t_wm)logp$ | $(t_s+t_wm)logp$ | $(t_s+t_wm)logp$ |
| All-to-one reduction | $(t_s+t_wm)logp$ | $(t_s+t_wm)logp$ | $(t_s+t_wm)logp$ |
| All-to-all broadcast | $(t_s + t_wm)(p-1)$ | $2t_s(\sqrt{p}-1) + t_wm(p-1)$ | $t_s(logp) + t_wm(p-1).$ |
| All-to-all reduction | $(t_s + t_wm)(p-1)$ | $2t_s(\sqrt{p}-1) + t_wm(p-1)$ | $t_s(logp) + t_wm(p-1).$ |
| All-reduce | $(t_s + t_wm)(p-1)$ | $2(\sqrt{p}-1)(ts + t_wm)$ | $(t_s + twm)logp$ |
| Pefix Sum | $(t_s + t_wm)(p-1)$ | $2(\sqrt{p}-1)(ts + t_wm)$ | $(t_s + twm)logp$ |
| Scatter | $t_s \log p + twm\,(p-1)$ | $t_s \log p + twm\,(p-1)$ | $t_s \log p + twm\,(p-1)$ |
| Gather | $t_s \log p + twm\,(p-1)$ | $t_s \log p + twm\,(p-1)$ | $t_s \log p + twm\,(p-1)$ |
| All-to-all personalized | $t_s+t_wmp/2(p-1)$ | $(2t_s + t_wmp)(\sqrt{p}-1)$ | $t_s+t_wmp/2(logp)$<br>Or $t_s+t_wm(p-1)$ |
| Circular Shift | $t_s+t_wm\ (p/2)$ | $t_s+t_wm\ (\sqrt{p}+1)$ | $t_s+t_wm(2logp-1)$<br>Or $t_s+t_wm$ |

# Basic Communication Operations in MPI


| Communication Operation | MPI Function |
|---|---|
| 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 |
| Prefix-Sum | MPI_Scan |
| Scatter | MPI_Scatter |
| Gather | MPI_Gather |
| All-to-All Personalized Communication | MPI_Alltoall |
| Circular Shift | |
| | |
| | |

# References



- Section 2.5 of text book
- Chapter 4 of text book.

innovate achieve lead

BITS Pilani
Pilani Campus

# Thank You