CZ4102 – High Performance Computing

Lecture 6:

Basic Communication Operations

- Dr Tay Seng Chuan

Reference: "Introduction to Parallel Computing" – Chapter 4.

Topic Overview

- One-to-All Broadcast and All-to-One Reduction
- All-to-All Broadcast and Reduction
- All-Reduce Operation
- Scatter and Gather

Basic Communication Operations: Introduction

- Many interactions in practical parallel programs occur in well-defined patterns involving groups of processors.
- Efficient implementations of these operations can improve performance, reduce development effort and cost, and improve software quality.
- Efficient implementations must leverage the underlying architecture. For this reason, we refer to specific architectures in this lecture such as linear array, ring, mesh and hypercube.

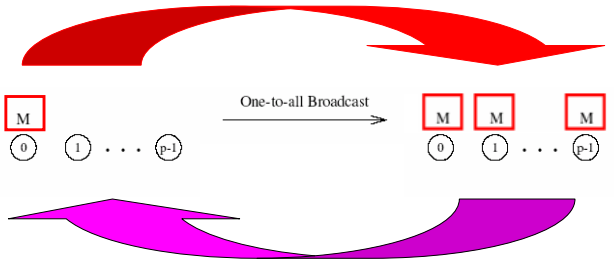
Basic Communication Operations: Introduction

- Group communication operations are built using point-to-point messaging primitives.
- Recall from our discussion of architectures that communicating a message of size m (ie, m words) over an uncongested network takes time $t_s + t_w m$ (setup time + transmission time for all words.)
- 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 (2 ways) and that communication is single-ported (one communication at one time).

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

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

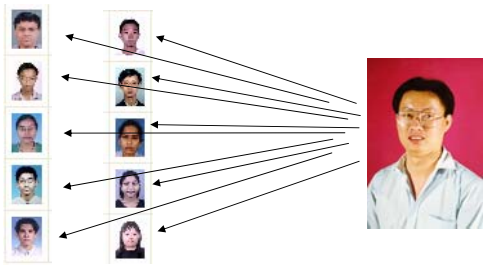
One-to-All Broadcast and All-to-One Reduction



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

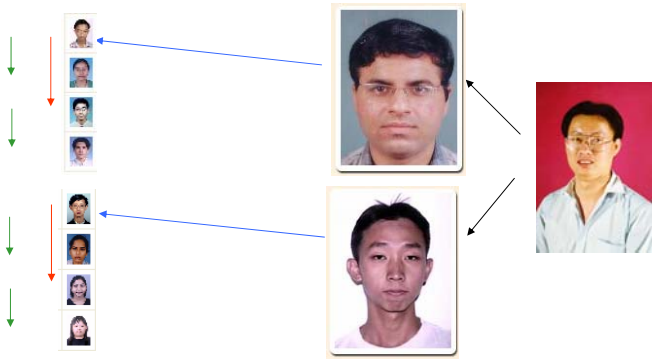
One-to-All Broadcast

- Simplest way is to send *p-1* messages from the source to the other *p-1* processors - this is not very efficient because there is no concurrency.



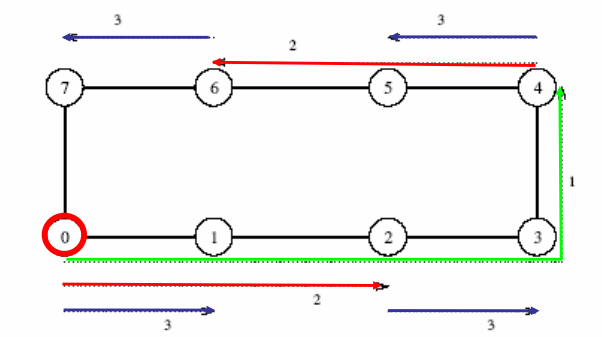
One-to-All Broadcast

- Use recursive doubling: source sends a message to a selected processor. We now have two independent problems derived over halves of machines.



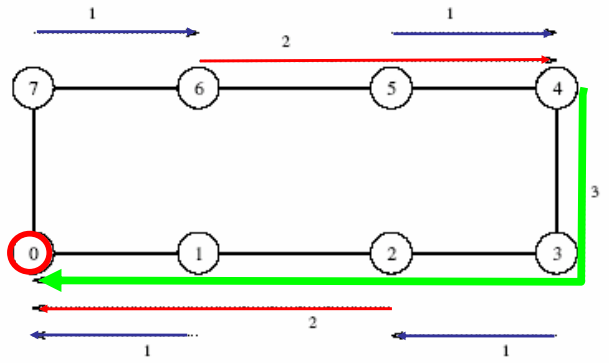
- Reduction can be performed in an identical fashion by inverting the process.

One-to-All Broadcast on a Ring



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

All-to-One Reduction on a Ring

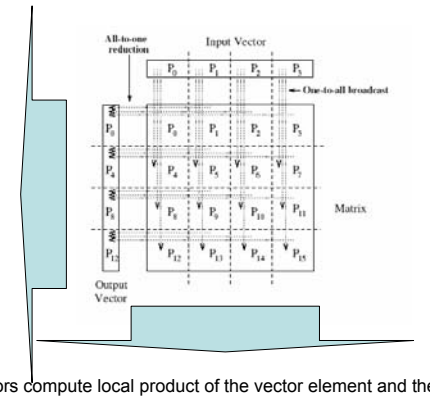


Reduction on an eight-node ring with node 0 as the destination of the reduction.

Broadcast and Reduction: Application Example

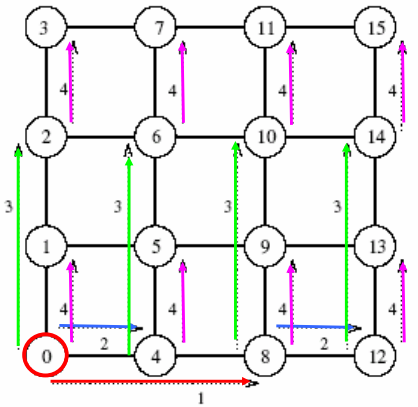
Consider the problem of multiplying a matrix (*n x n*) with a vector (*n x 1*).

- 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 column using *n* concurrent all-to-one reduction operations along the columns (using the sum operation).

Broadcast and Reduction on a Mesh

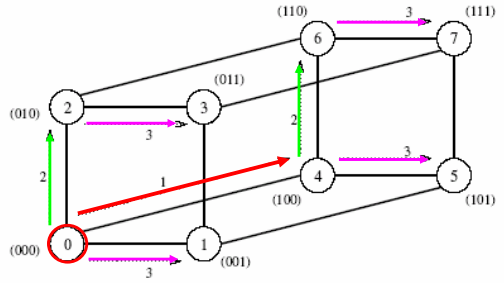


One-to-all broadcast on a 16-node mesh.

(This is a similar extension of the linear array counter-part.)

Broadcast and Reduction on a Hypercube

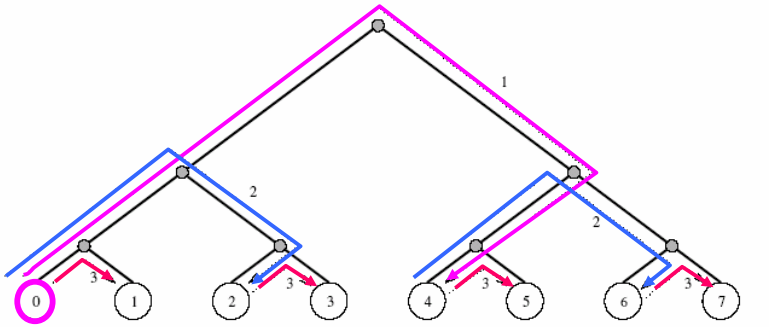
- A hypercube with 2^d nodes can be regarded as a d-dimensional mesh with two nodes in each dimension.
- The mesh algorithm can be generalized to a hypercube and the operation is carried out in d (= $\log p$) steps.



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

Broadcast and Reduction on a Balanced Binary Tree

- Consider a binary tree in which processors are (logically) at the leaves and internal nodes are routing nodes.
- Assume that source processor is the left most node. The process of broadcast is done in progressive manner. Reduction is done in reversed order.



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

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 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.

Logical Operators Used in the Algorithm

XOR	0	1
0	0	1
1	1	0

One and only one.

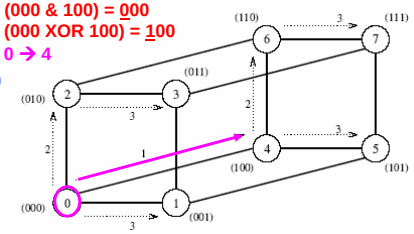
AND	0	1
0	0	0
1	0	1

All must be 1 (all must be true.)

A mask is used to inspect a particular segment of a bit pattern.

One-to-All Broadcast Algorithm on Hypercube (Case: 100: 0 to 4)

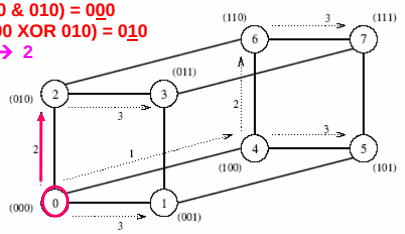
```
1.  procedure ONE_TO_ALL_BC(d, my_id, X)
2.  begin
3.      mask := 2^d − 1;                    /* Set all d bits of mask to 1 */   (111)
4.      for i := d − 1 downto 0 do          /* Outer loop */   i = d-1 = 2
5.          mask := mask XOR 2^i;           /* Set bit i of mask to 0 */   111 XOR 100 = (011)
(100 & 011) = 000   if (my_id AND mask) = 0 then   /* If lower i bits of my_id are 0 */   (000 & 011) = 000
(100 & 100) = 100       if (my_id AND 2^i) = 0 then                                   (000 & 100) = 000
8.                          msg_destination := my_id XOR 2^i;                         (000 XOR 100) = 100
9.                          send X to msg_destination;                                0 → 4
10.                     else                                                          (100 XOR 100) = 000
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
```



Algorithm 4.1 One-to-all broadcast of a message X from node 0 of a d-dimensional p-node hypercube ($d = \log p$). AND and XOR are bitwise logical-and and exclusive-or operations, respectively.

One-to-All Broadcast Algorithm on Hypercube Case: (010: 0 to 2)

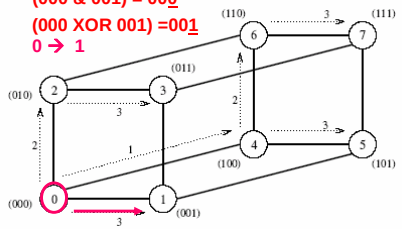
```
1.  procedure ONE_TO_ALL_BC(d, my_id, X)
2.  begin
3.      mask := 2^d − 1;                    /* Set all d bits of mask to 1 */   (111)
4.      for i := d − 1 downto 0 do          /* Outer loop */   i=1
5.          mask := mask XOR 2^i;           /* Set bit i of mask to 0 */   111 XOR 010 = (101)
(010 & 101) = 000   if (my_id AND mask) = 0 then   /* If lower i bits of my_id are 0 */   (000 & 101) = 000
(010 & 010) = 010       if (my_id AND 2^i) = 0 then                                   (000 & 010) = 000
8.                          msg_destination := my_id XOR 2^i;                         (000 XOR 010) = 010
9.                          send X to msg_destination;                                0 → 2
10.                     else                                                          (010 XOR 010) = 000
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
```



Algorithm 4.1 One-to-all broadcast of a message X from node 0 of a d-dimensional p-node hypercube ($d = \log p$). AND and XOR are bitwise logical-and and exclusive-or operations, respectively.

One-to-All Broadcast Algorithm on Hypercube Case: (001 : 0 to 1)

```
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 */   (111)
4.      for i := d − 1 downto 0 do          /* Outer loop */   i=0
5.          mask := mask XOR 2^i;           /* Set bit i of mask to 0 */   111 XOR 001 = (110)
(001 & 110) = 000   if (my_id AND mask) = 0 then   /* If lower i bits of my_id are 0 */   (000 & 110) = 000
(001 & 001) = 001       if (my_id AND 2^i) = 0 then                                   (000 & 001) = 000
8.                          msg_destination := my_id XOR 2^i;                         (000 XOR 001) =001
9.                          send X to msg_destination;                                0 → 1
10.                     else                                                          (001 XOR 001) = 000
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
```

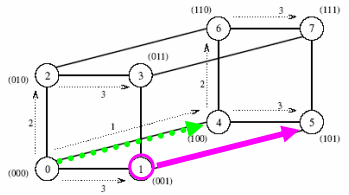


Algorithm 4.1 One-to-all broadcast of a message X from node 0 of a d-dimensional p-node hypercube ($d = \log p$). AND and XOR are bitwise logical-and and exclusive-or operations, respectively.

Broadcast and Reduction Algorithms

```
1.  procedure GENERAL_ONE_TO_ALL_BC(d, my_id, source, X)
2.  begin
(101 XOR    my_virtual_id := my_id XOR source;                    (001 XOR 001) = 000
001) = 100  mask := 2^d − 1;                                      (111)
            for i := d − 1 downto 0 do    /* Outer loop */        i=2
6.              mask := mask XOR 2^i;     /* Set bit i of mask to 0 */   111 XOR 100 = (011)
(100 & 011) = 000   if (my_virtual_id AND mask) = 0 then          (000 & 011) = 000
(100 & 100) = 100       if (my_virtual_id AND 2^i) = 0 then       (000 & 100) = 000
9.                          virtual_dest := my_virtual_id XOR 2^i;   (000 XOR 100) = 100
10.                         send X to (virtual_dest XOR source);     (100 XOR 001) = 101  (5)
        /* Convert virtual_dest to the label of the physical destination */
11.                     else
12.                         virtual_source := my_virtual_id XOR 2^i;   (100 XOR 100) = 000
13.                         receive X from (virtual_source XOR source);   (000 XOR 001) = 001
        /* 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.



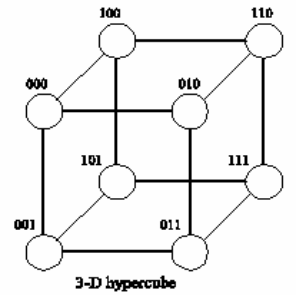
Cost Analysis for One-to-Many Broadcast and Many-to-One Reduction on Hypercube

- The broadcast or reduction procedure involves *log p* (where *log p* is the dimension) point-to-point simple message transfers, each at a time cost of

  $t_s + t_w m$ *(setup time + transmission time for all words)*.

- The total time is therefore given by:

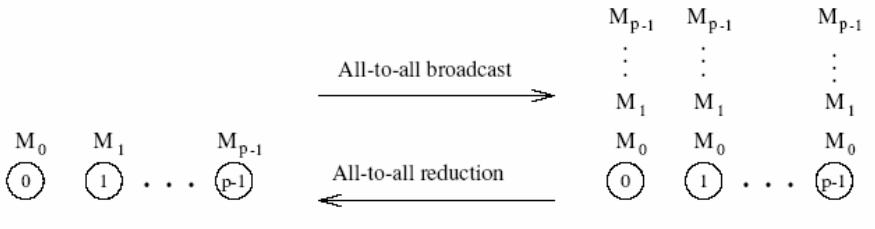
$$T = (t_s + t_w m) \log p.$$



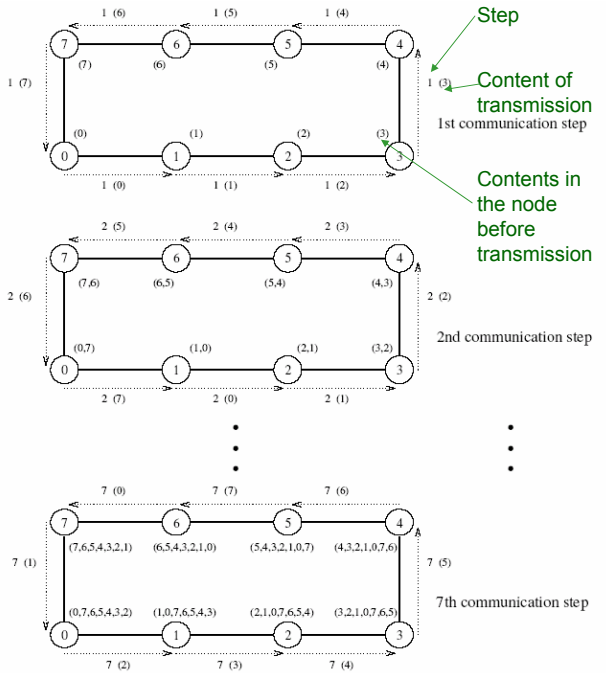
All-to-All Broadcast and Reduction

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

All-to-All Broadcast and Reduction



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



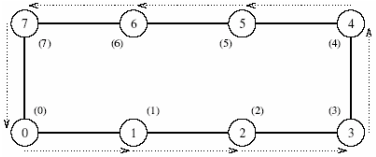
All-to-All Broadcast and Reduction on a Ring

- Simplest approach: perform *p* one-to-all broadcasts. This is not the most efficient way.
- Each node first sends to one of its neighbours the data it needs to broadcast.
- In subsequent steps, it forwards the data received from one of its neighbors to its other neighbor.
- The algorithm terminates in *p-1* steps.

All-to-All Broadcast and Reduction on a Ring

```
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
```

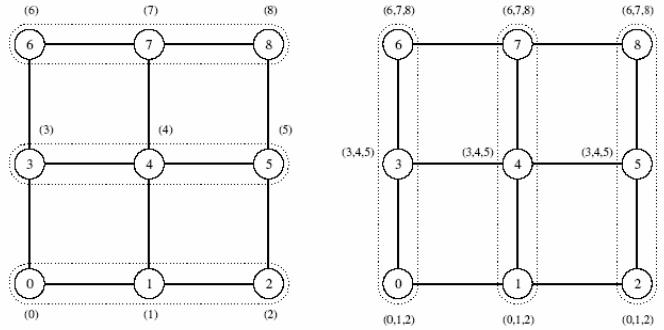
mod: remainder after the division by *p*



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

All-to-all Broadcast on a Mesh

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



(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).

translate to left end

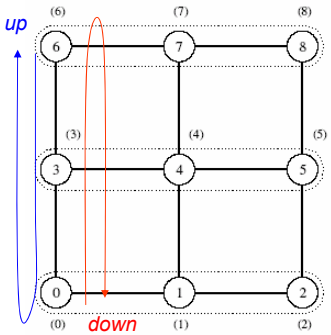
All-to-all Broadcast on a Mesh

```
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
```

add the offset

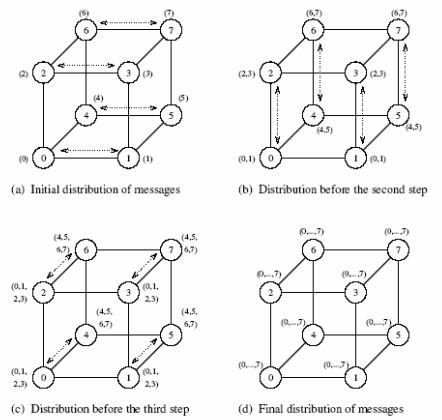
(0-3)%9 = 6

(0+3)%9 = 3



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

All-to-all broadcast on a Hypercube



(a) Initial distribution of messages (b) Distribution before the second step

(c) Distribution before the third step (d) Final distribution of messages

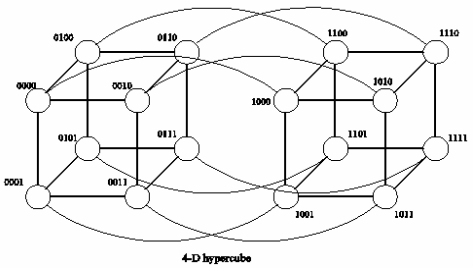
- Generalization of the mesh algorithm to *log p* dimensions.
- Message size doubles at each of the *log p* steps.

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

All-to-all broadcast on a Hypercube

```
procedure ALL_TO_ALL_BC_HCUBE(my_id, my_msg, d, result)
begin
    result := my_msg;
    for i := 0 to d − 1 do
        partner := my_id XOR 2^i;
        send result to partner;
        receive msg from partner;
        result := result ∪ msg;
    endfor;
end ALL_TO_ALL_BC_HCUBE
```

```
    a b c d e f g h
xor 0 0 0 1 0 0 0 0
---------------------
    a b c d e f g h
```



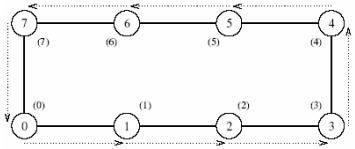
All-to-all broadcast on a *d*-dimensional hypercube.

All-to-all Reduction

- Similar communication pattern to all-to-all broadcast, except in the reverse order.
- On receiving a message, a node must combine it with the local copy of the message that has the same destination as the received message before forwarding the combined message to the next neighbor.

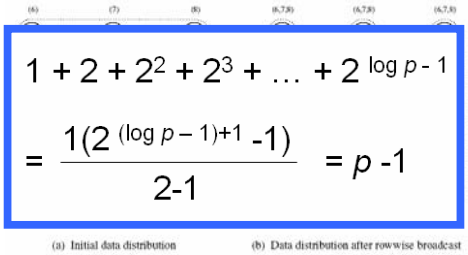
Cost Analysis for All-to-All Broadcast and Reduction

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



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

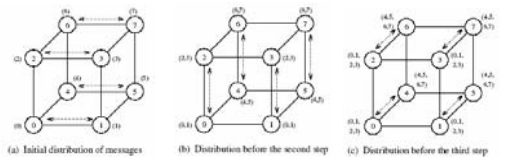
$$1 + 2 + 2^2 + 2^3 + \ldots + 2^{\log p - 1} = \frac{1(2^{(\log p - 1)+1} - 1)}{2-1} = p - 1$$



- On a hypercube, we have:

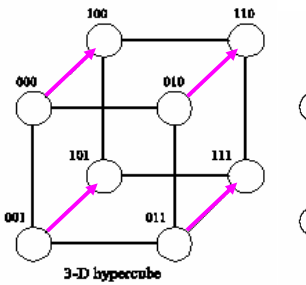
$$T = \sum_{i=1}^{\log p} (t_s + 2^{i-1} t_w m)$$

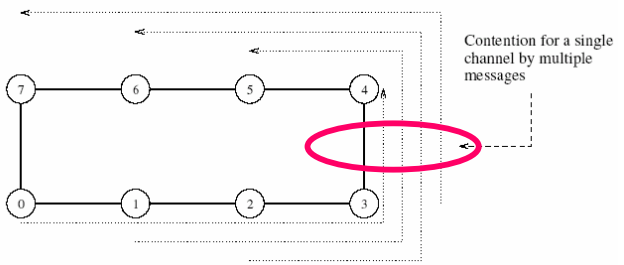
$$= t_s \log p + t_w m(p-1).$$



All-to-all broadcast: Notes

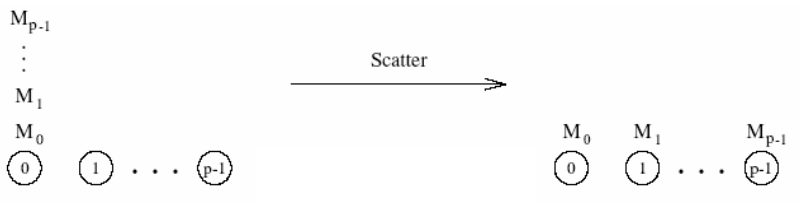
- All of the algorithms presented above are asymptotically optimal in message size.
- It is not possible to port algorithms for higher dimensional networks (such as a hypercube) into a ring because this would cause **contention**.





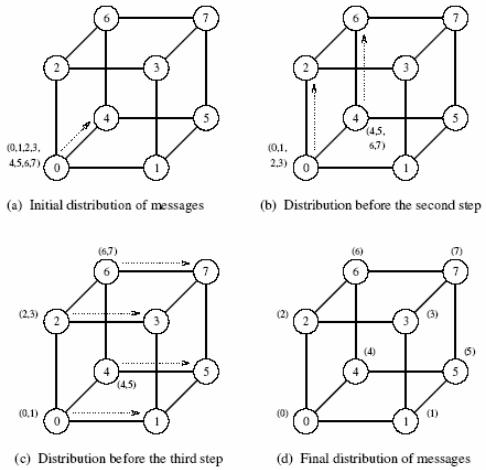
Scatter and Gather

- 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, ie, different node will receive different contents).
- In the *gather* operation, a single node collects a unique message from each node.



- While the scatter operation is fundamentally different from broadcast (all nodes receive the same content), 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.

Example of the Scatter Operation on Hypercube



The scatter operation on an eight-node hypercube.

Cost of Scatter and Gather for Hypercube

- There are *log p* steps. In each step, the machine size doubles and the data size halves.
- We have the time for this operation to be (refer to summation in All-to-All Broadcast):

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

- This time holds for a linear array as well as a 2-D mesh (tutorial questions).