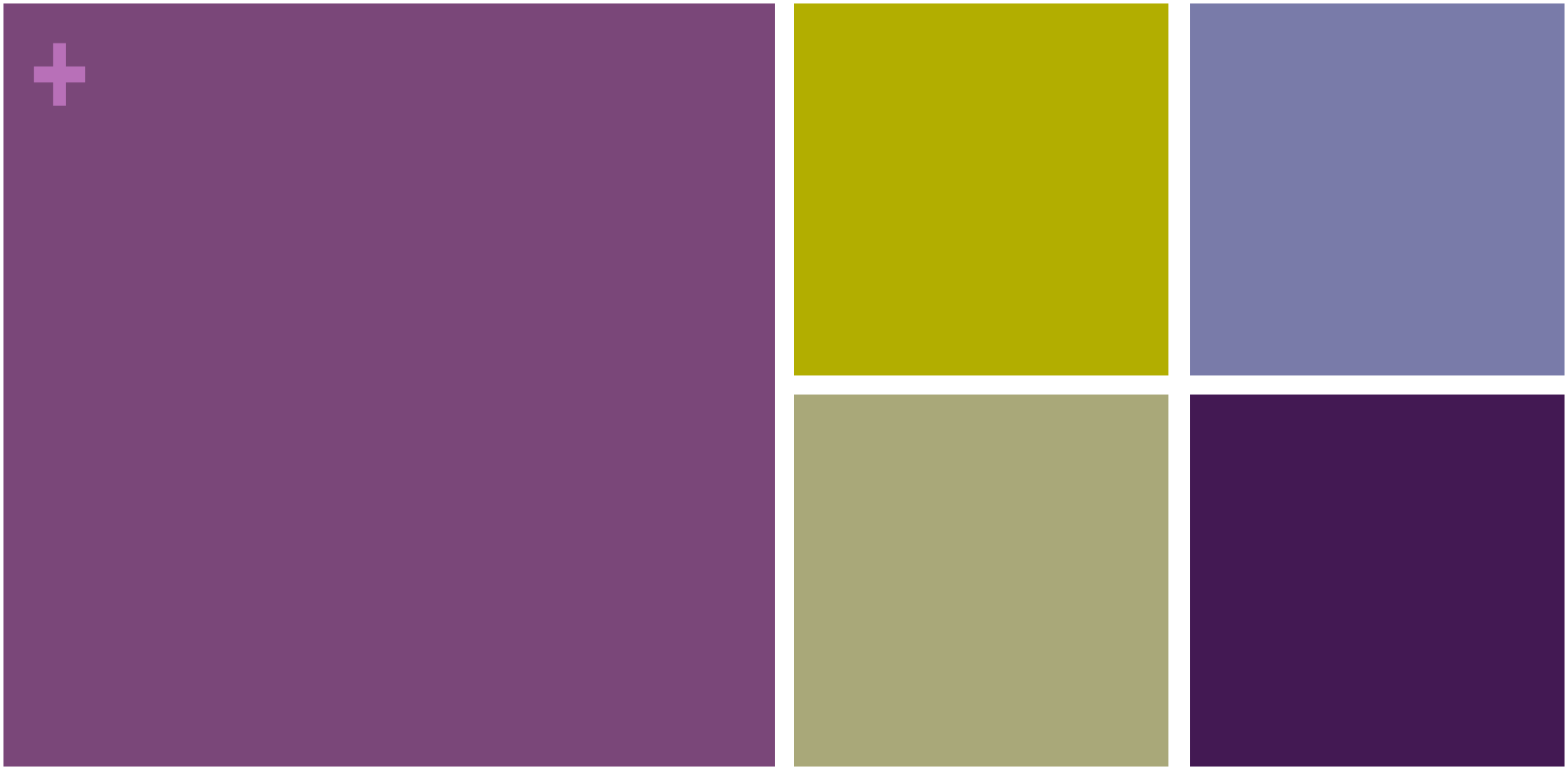


Design of Parallel Algorithms

Communication Algorithms

Topic Overview

- One-to-All Broadcast and All-to-One Reduction
- All-to-All Broadcast and Reduction
- All-Reduce and Prefix-Sum Operations
- Scatter and Gather
- All-to-All Personalized Communication
- Improving the Speed of Some Communication Operations

Basic Communication Operations: Introduction

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

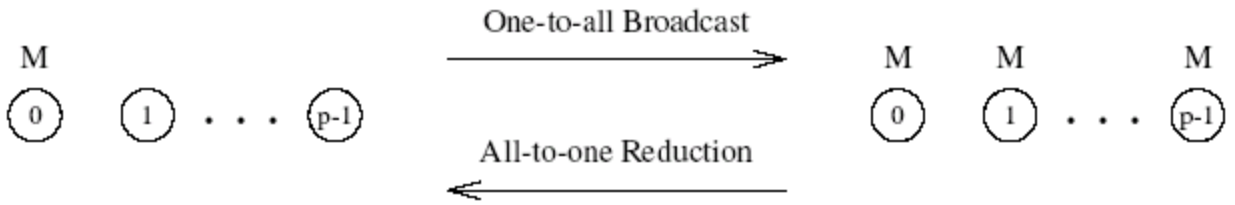
Basic Communication Operations: Introduction

- Group communication operations are built using point-to-point messaging primitives.
- Recall from our discussion of architectures that communicating a message of size m over an 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 and that communication is single-ported.

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

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

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

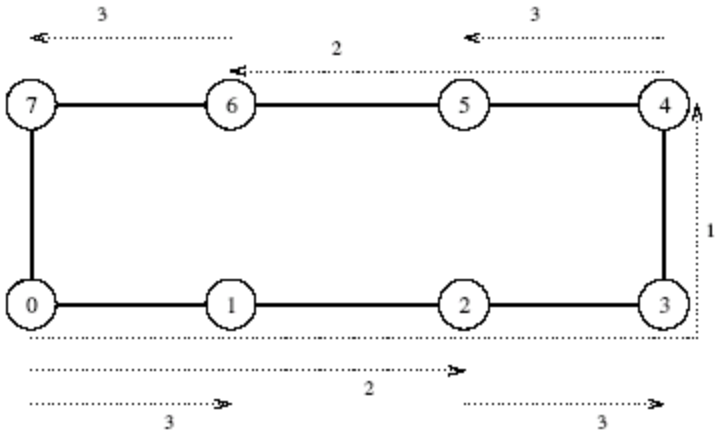


One-to-all broadcast and all-to-one reduction among processors.

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

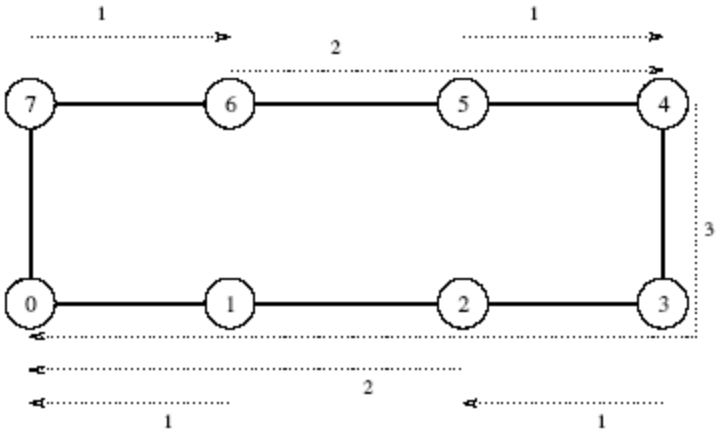
- Simplest way is to send *p-1* messages from the source to the other *p-1* processors - this is not very efficient.
- Use recursive doubling: source sends a message to a selected processor. We now have two independent problems derined over halves of machines.
- Reduction can be performed in an identical fashion by inverting the process.

One-to-All Broadcast



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

All-to-One Reduction



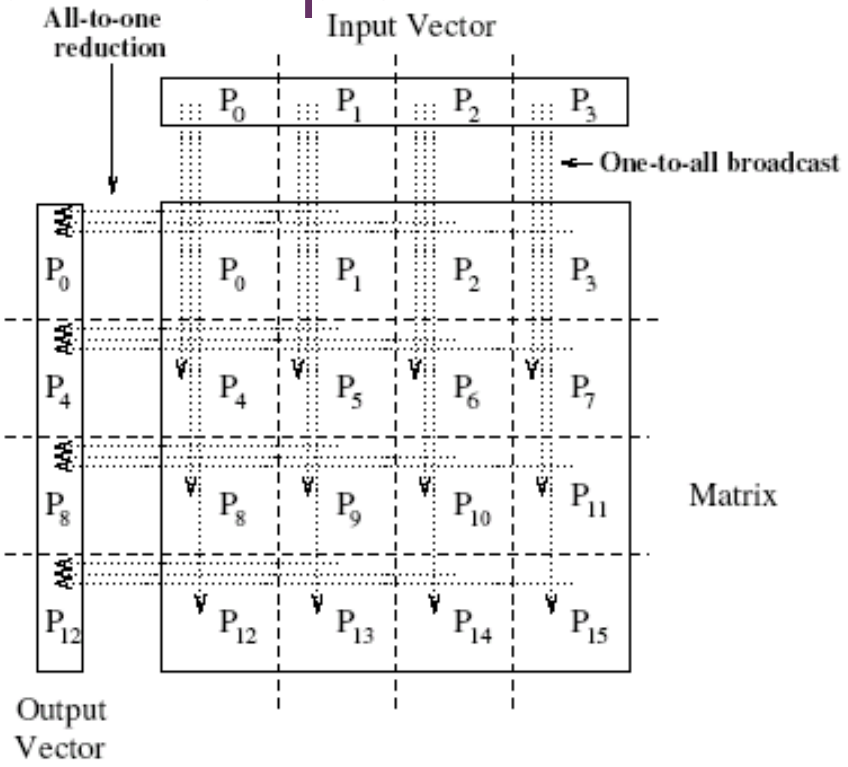
Reduction on an eight-node ring with node 0 as the destination of the reduction.

Broadcast and Reduction: Example

Consider the problem of multiplying a matrix with a vector.

- The *n 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

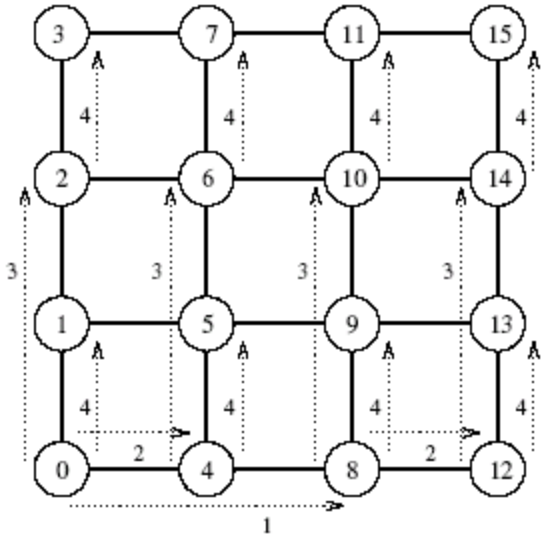


One-to-all broadcast and all-to-one reduction in the multiplication of a *4 x 4* matrix with a *4 x 1* vector.

Broadcast and Reduction on a Mesh

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

Broadcast and Reduction on a Mesh: Example

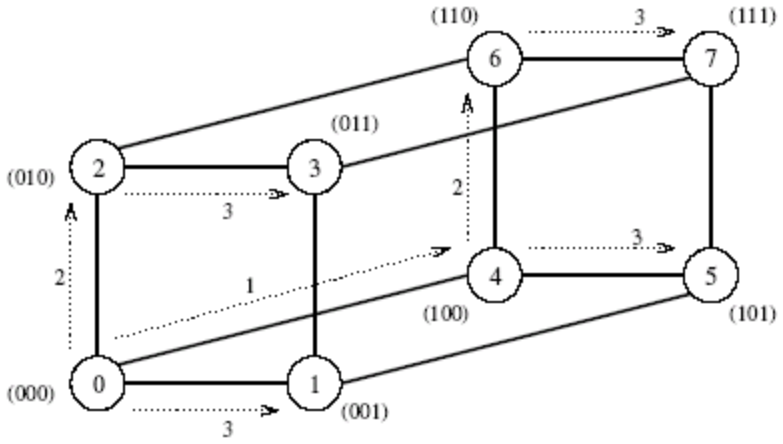


One-to-all broadcast on a 16-node mesh.

Broadcast and Reduction on a Hypercube

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

Broadcast and Reduction on a Hypercube: Example



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

Broadcast and Reduction 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.
- An algorithm to broadcast from 0 is simply implemented by utilizing how the address bits map to the recursive construction of the hypercube
- To support arbitrary source processors we us a mapping from physical processors to virtual processors. We always send from processor 0 in the virtual processor space.
- The XOR operation with the root gives us a idempotent mapping operation (apply once to get from virtual->physical, second time to get from physical->virtual)
- Pseudo code in this chapter assumes buffered communication! Must modify appropriately to make correct MPI implementations.

Broadcast and Reduction Algorithms

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

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

Broadcast and Reduction Algorithms

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

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

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:

$$T_{comm} = \sum_{i=1}^{\log p} \left(t_s + t_w m\right) = \left(t_s + t_w m\right) \log p$$

Useful Identities for analysis of more complex algorithms to come

- Geometric Series:

$$\sum_{k=1}^{n} r^k = \frac{r\left(r^n - 1\right)}{r-1} \Rightarrow \sum_{i=1}^{\log p} 2^{i-1} = p - 1$$

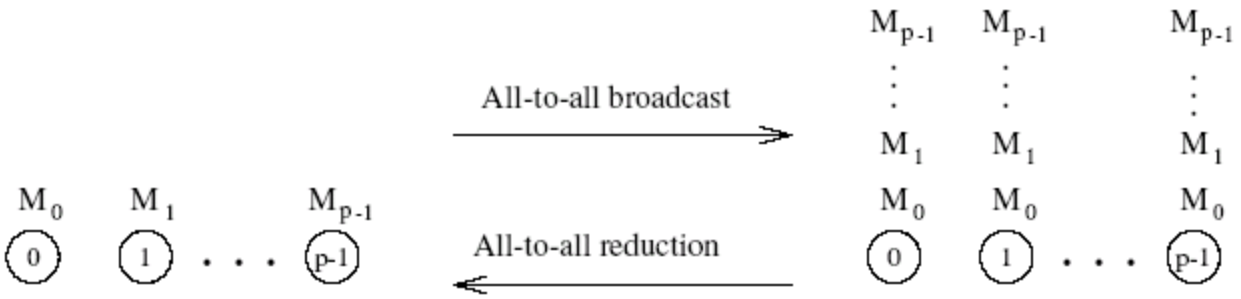
- Euler's Identity:

$$\sum_{k=1}^{n} k = \frac{n\left(n+1\right)}{2}$$

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

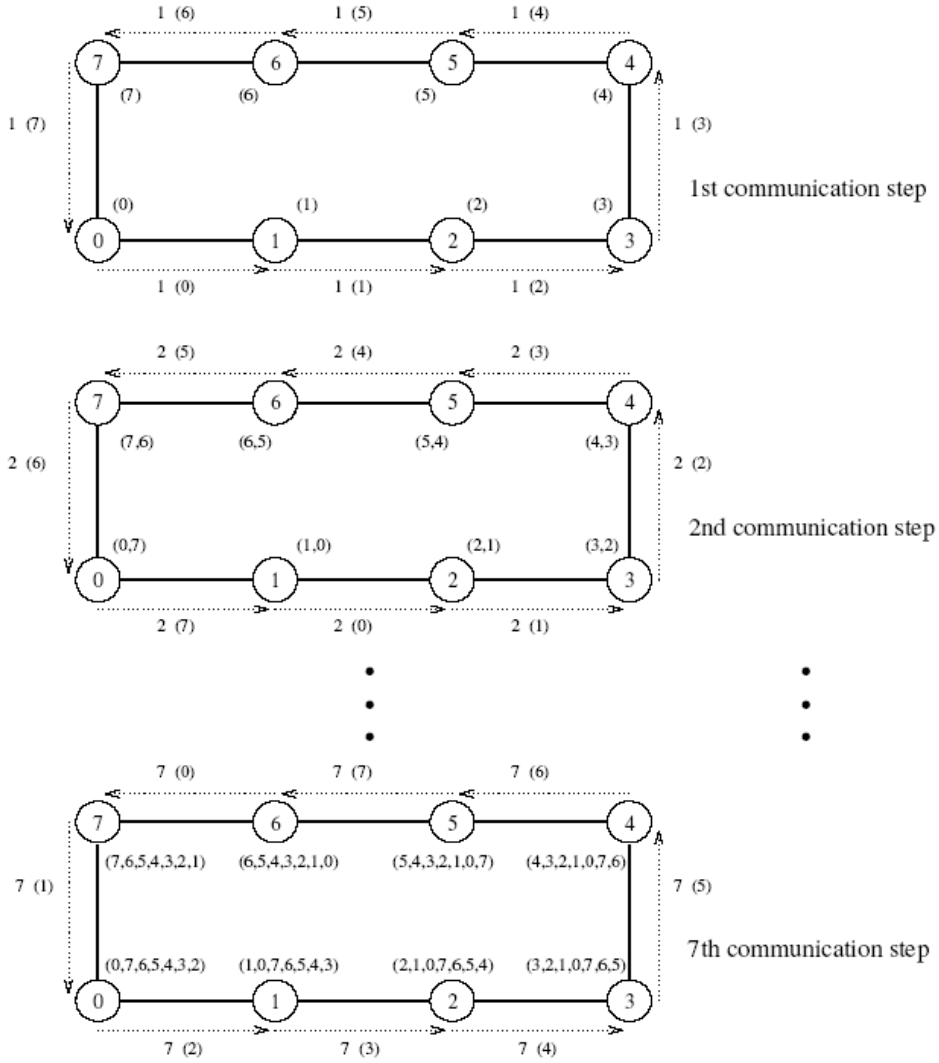


All-to-all broadcast and all-to-all reduction.

All-to-All Broadcast and Reduction on a Ring

- Can be thought of as a one-to-all broadcast where every processor is a root node
- Naïve implementation: perform *p* one-to-all broadcasts. This is not the most efficient as processors often idle waiting for messages to arrive in each independent broadcast.
- A better way can perform the operation in p steps:
  - Each node first sends to one of its neighbors the data it needs to broadcast.
  - In subsequent steps, it forwards the data received from one of its neighbors to its other neighbor.
  - The algorithm terminates in *p-1* steps.

All-to-All Broadcast and Reduction on a Ring



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

All-to-All Broadcast 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
```

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

Analysis of ring all-to-all broadcast algorithm

- The algorithm does p-1 steps and in each step it sends and receives a message of size m.
- Therefore the communication time is:

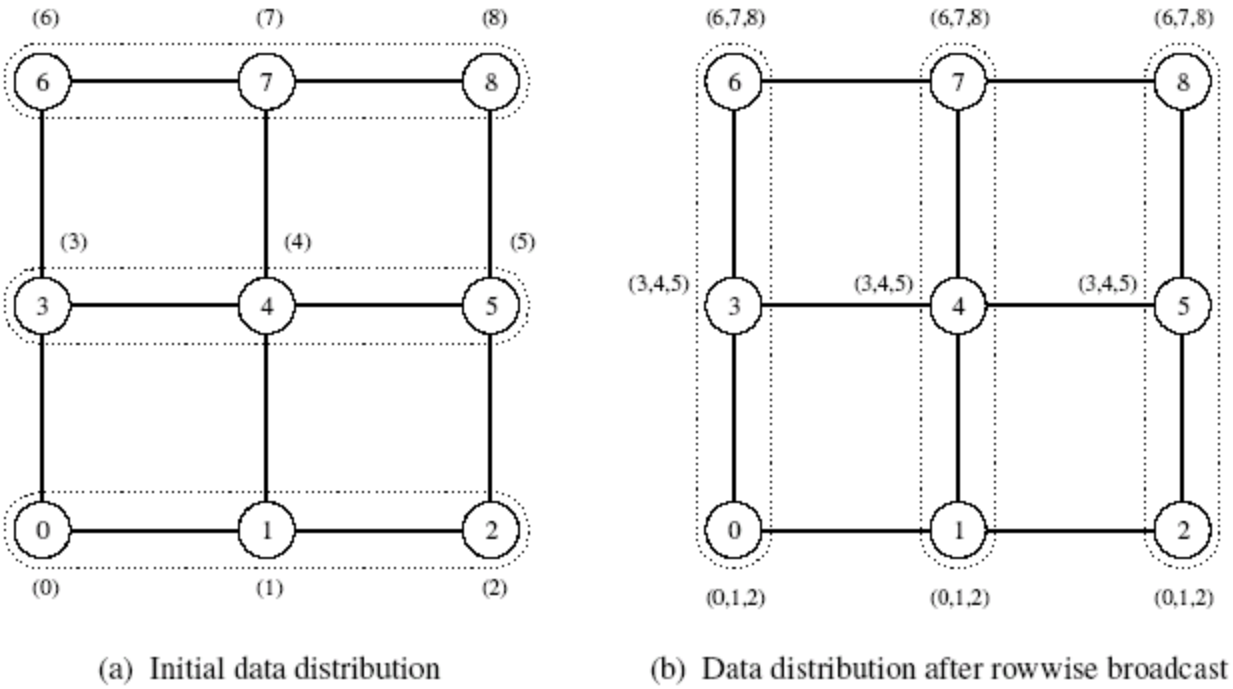
$$T_{all-to-all-ring} = \sum_{i=1}^{p-1} \left(t_s + t_w m\right) = (t_s + t_w m)(p-1)$$

- Note that the bisection width of the ring is 2, while the communication pattern requires the transmission of p/2 pieces of information from one half of the network to the other. Therefore the all-to-all broadcast cannot be faster than O(p) for a ring. Therefore this algorithm is asymptotically optimal.

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 column-wise all-to-all broadcast of the consolidated messages.

All-to-all Broadcast on a Mesh



(a) Initial data distribution

(b) Data distribution after rowwise broadcast

All-to-all broadcast on a *3 x 3* mesh. The groups of nodes communicating with each other in each phase are enclosed by dotted boundaries. By the end of the second phase, all nodes get (0,1,2,3,4,5,6,7) (that is, a message from each node).

All-to-all Broadcast on a Mesh

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

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

Mesh based All-to-All broadcast Analysis

- Algorithm proceeds in two steps: 1) ring broadcast over rows with message size = *m*, then ring broadcast over columns with message size = $\sqrt{p}\,m$
- Time for communication:

$$T_{comm} = \overbrace{\left(t_s + t_w m\right)\left(\sqrt{p}-1\right)}^{step1} + \overbrace{\left(t_s + t_w \sqrt{p} m\right)\left(\sqrt{p}-1\right)}^{step2}$$

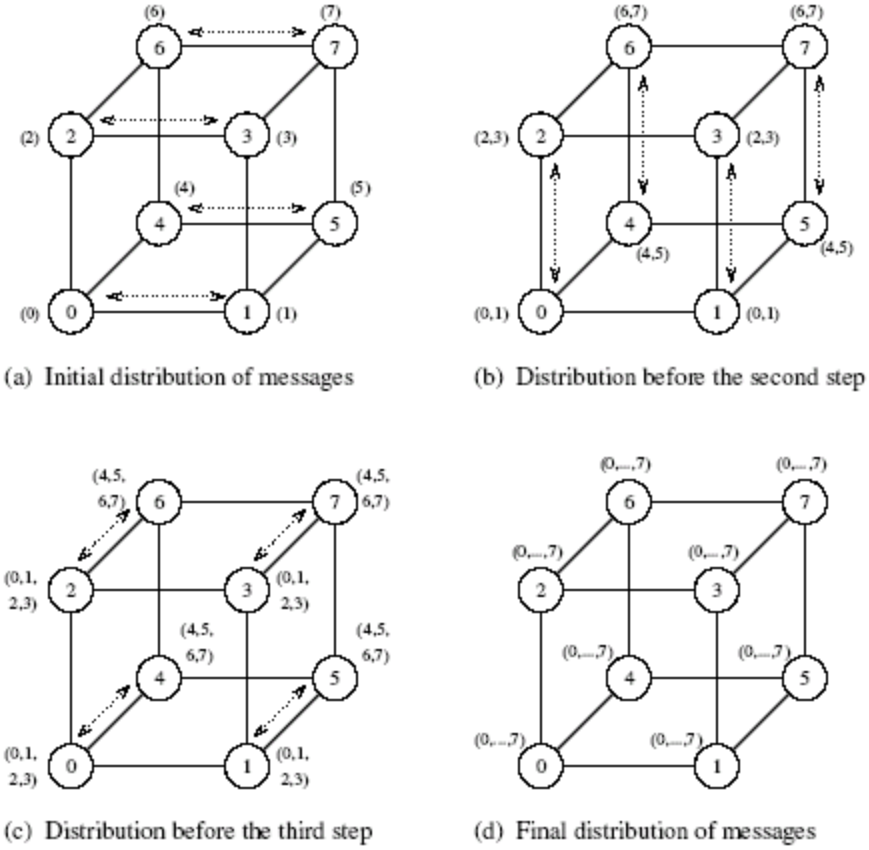
$$T_{comm} = 2t_s\left(\sqrt{p}-1\right) + t_w m\left(p-1\right)$$

- Due to single-port assumption, all-to-all broadcast cannot execute faster than O(p) time since each processor must receive p-1 distinct messages. Therefore this algorithms is asymptotically optimal.

All-to-all broadcast on a Hypercube

- Generalization of the mesh algorithm to *log p* dimensions.
- Message size doubles at each of the *log p* steps.
  - Note: analysis of this algorithm will utilize geometric series identity due to the doubling messages sizes

All-to-all broadcast on a Hypercube



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

All-to-all broadcast on a Hypercube

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

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

All-to-all Reduction

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

Cost Analysis
All-to-all communication

- On a ring, the time is given by: $T_{ring} = (t_s + t_w m)(p-1)$
- On a mesh, the time is given by: $T_{mesh} = 2t_s\left(\sqrt{p}-1\right) + t_w m(p-1)$
- On a hypercube, we have:

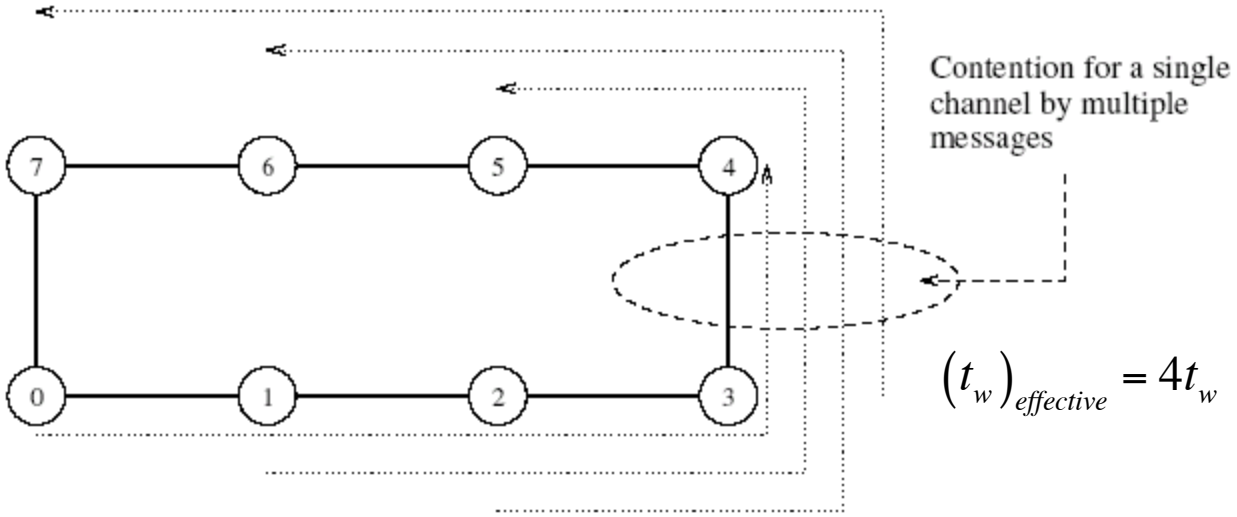
$$T_{hypercube} = \sum_{i=1}^{\log p}\left(t_s + 2^{i-1} t_w m\right)$$

$$T_{hypercube} = t_s \log p + t_w m(p-1)$$

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 network congestion.
  - We are utilizing a network model whereby we know that we can get full link bandwidth at every step of the algorithm because the communication pattern maps onto the network with every link having exclusive use for a single communication
  - If we were to map the algorithm onto a lower dimensional network, we would need to multiply the t_w term by the number of messages sharing the link to account for the effect of link congestion

All-to-all broadcast: Notes



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

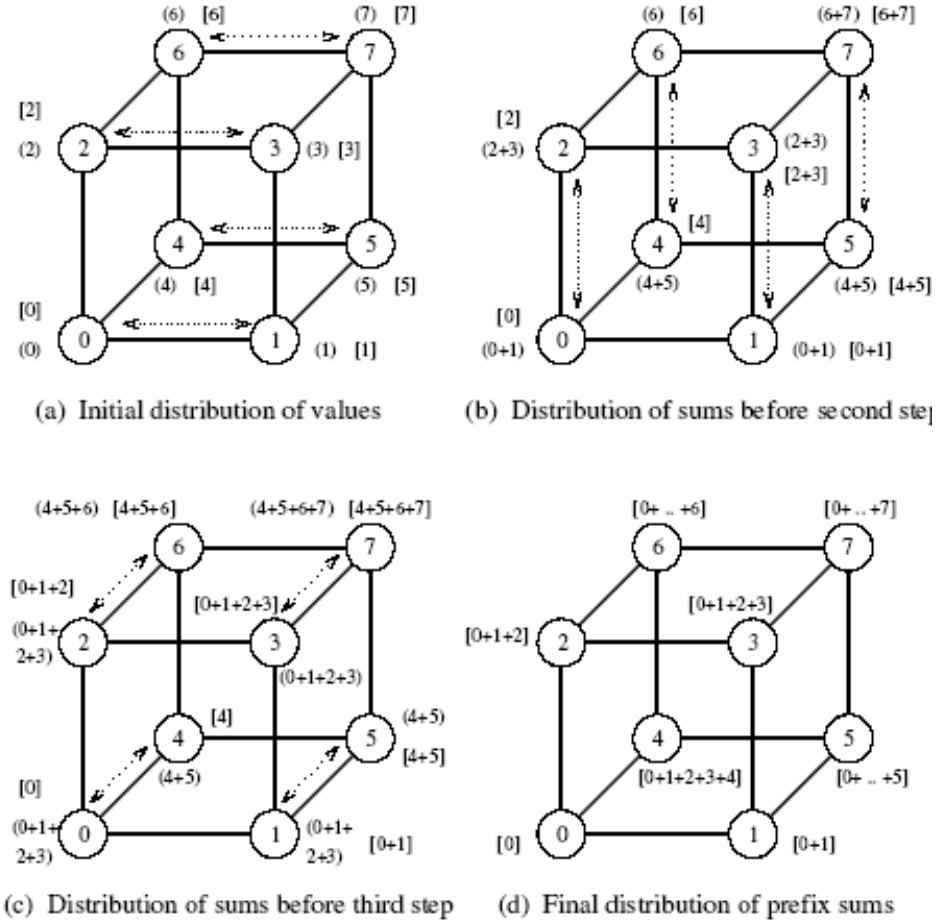
All-Reduce and Prefix-Sum Operations

- In all-reduce, each node starts with a buffer of size *m* and the final results of the operation are identical buffers of size *m* on each node that are formed by combining the original *p* buffers using an associative operator.
- Identical to all-to-one reduction followed by a one-to-all broadcast. This formulation is not the most efficient. Uses the pattern of all-to-all broadcast, instead. The only difference is that message size does not increase here. Time for this operation is $(t_s + t_w m) \log p$ which is half the time of doing the two step implementation.
- Different from all-to-all reduction, in which *p* simultaneous all-to-one reductions take place, each with a different destination for the result.

The Prefix-Sum Operation

- 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.
- Very useful operation in determining the layout of distributed arrays:
  - Every processor has n_i elements that are numbered locally from $0, 1, \ldots, n_i$
  - A prefix sum is used to determine the global numbering when all of the local arrays are merged together to represent one unified, but distributed, array

The Prefix-Sum Operation



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

The Prefix-Sum Operation

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

The Prefix-Sum Operation

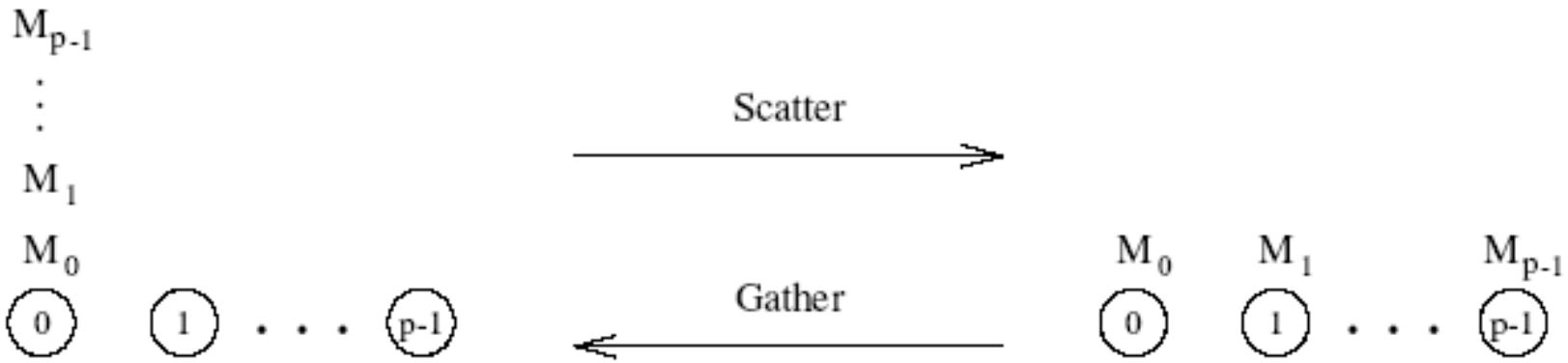
```
1.   procedure PREFIX_SUMS_HCUBE(my_id, my_number, d, result)
2.   begin
3.       result := my_number;
4.       msg := result;
5.       for i := 0 to d − 1 do
6.           partner := my_id XOR 2^i;
7.           send msg to partner;
8.           receive number from partner;
9.           msg := msg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

Prefix sums on a *d*-dimensional hypercube.

Scatter and Gather

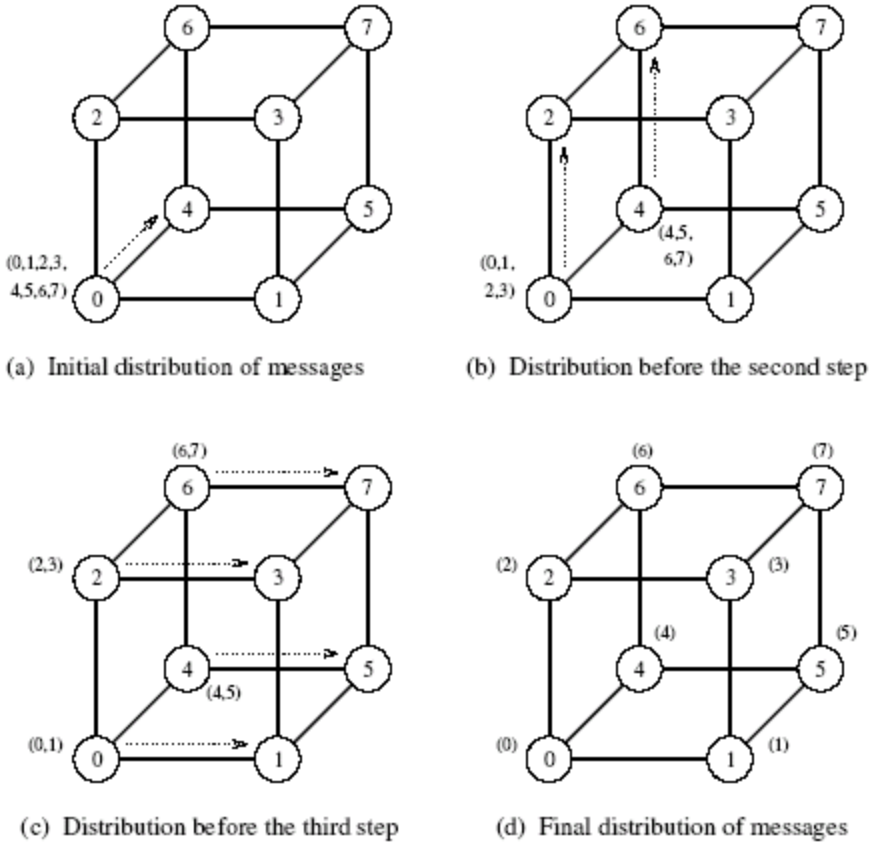
- In the *scatter* operation, a single node sends a unique message of size *m* to every other node (also called a one-to-all personalized communication).
- In the *gather* operation, a single node collects a unique message from each node.
- While the scatter operation is fundamentally different from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

Gather and Scatter Operations



Scatter and gather operations.

Example of the Scatter Operation



The scatter operation on an eight-node hypercube.

Cost of Scatter and Gather

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

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

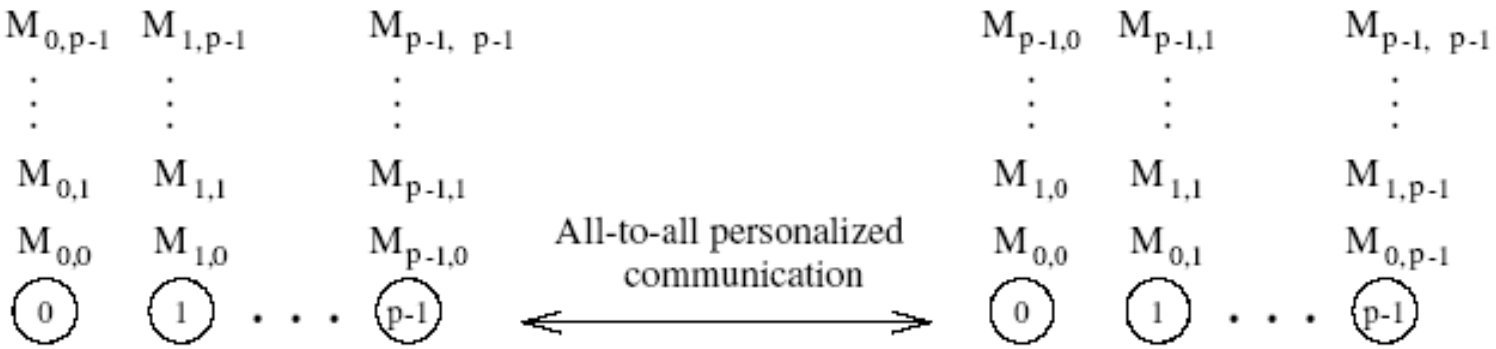
$$T = t_s \log p + t_w m \left(p - 1 \right)$$

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

All-to-All Personalized Communication

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

All-to-All Personalized Communication

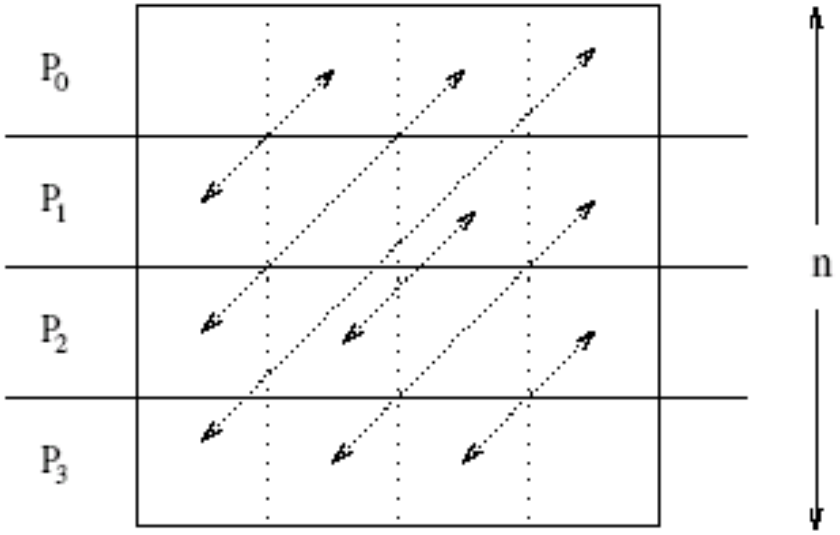


All-to-all personalized communication.

All-to-All Personalized Communication: Example

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

All-to-All Personalized Communication: Example

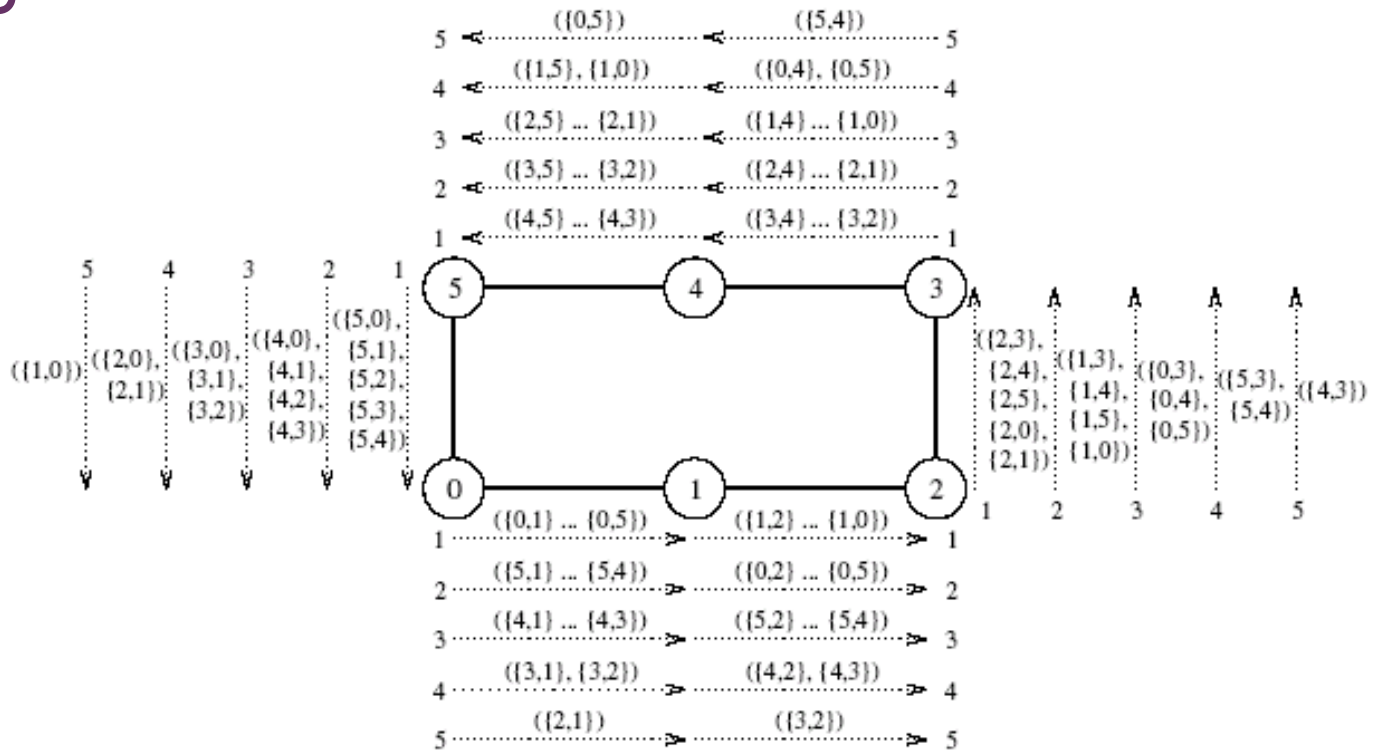


All-to-all personalized communication in transposing a *4 x 4* matrix using four processes.

All-to-All Personalized Communication on a Ring

- Each node sends all pieces of data as one consolidated message of size $m(p - 1)$ to one of its neighbors.
- Each node extracts the information meant for it from the data received, and forwards the remaining $(p - 2)$ pieces of size m each to the next node.
- The algorithm terminates in $p - 1$ steps.
- The size of the message reduces by m at each step.

All-to-All Personalized Communication on a Ring



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

$$T_{comm} = \sum_{i=1}^{p-1}\left(t_s + (p-i)t_w m\right) = \overbrace{\sum_{i=1}^{p-1}\left(t_s + it_w m\right)}^{\text{reorder sum}}$$

$$T_{comm} = t_s(p-1) + t_w m\sum_{i=1}^{p-1} i$$

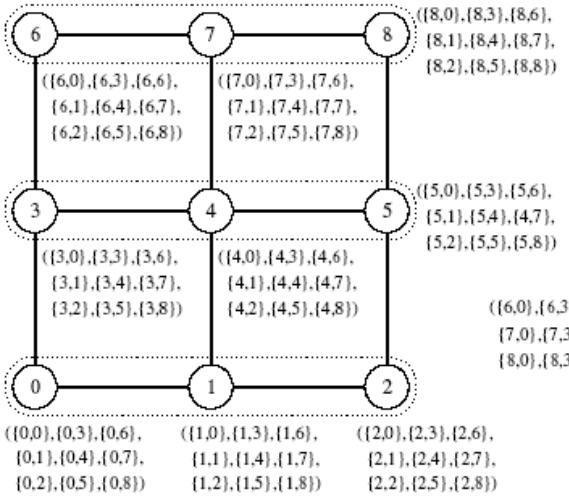
$$T_{comm} = \left(t_s + t_w m p/2\right)(p-1)$$

- Note, a ring has a bisection width of 2 while the all-to-all personalized communication algorithm will need to communicate $mp^2/2$ data between the bisections giving an asymptotic optimal time for this algorithm of $O(mp^2)$. This algorithm is asymptotically optimal.

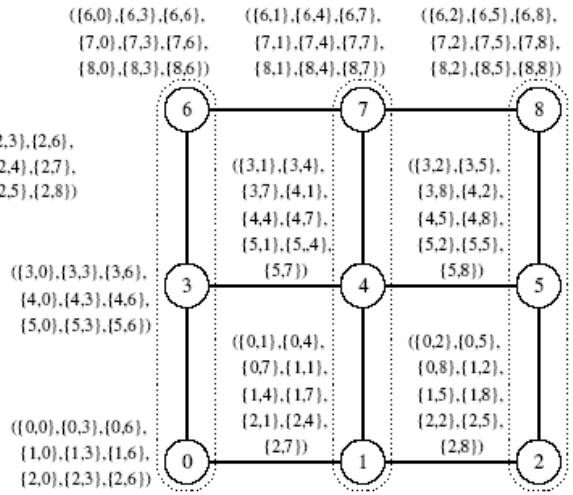
All-to-All Personalized Communication on a Mesh

- 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) Data distribution at the beginning of first phase



(b) Data distribution at the beginning of second phase

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

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

$$T_{comm} = 2\overbrace{\left(t_s + t_w m p/2\right)\left(\sqrt{p}-1\right)}^{ring\ \sqrt{p}\ procesors}$$

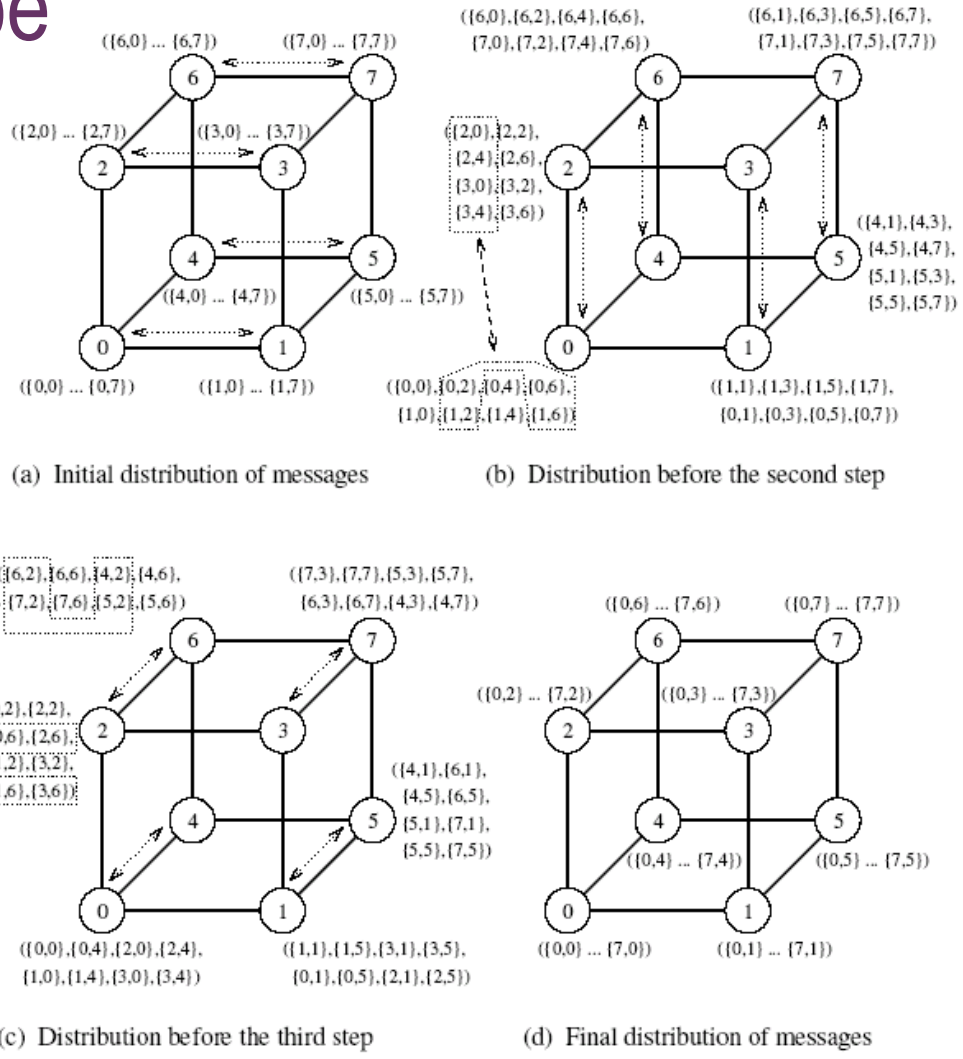
$$T_{comm} = \left(2t_s + t_w m p\right)\left(\sqrt{p}-1\right)$$

- Bisection width of the 2-D mesh is $O(\sqrt{p})$, therefore the fastest time to communicate $mp^2/2$ pieces of information between bisections is $O(mp\sqrt{p})$. This algorithm is asymptotically optimal for a 2-D mesh network.

All-to-All Personalized Communication on a Hypercube

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

All-to-All Personalized Communication on a Hypercube



(a) Initial distribution of messages

(b) Distribution before the second step

(c) Distribution before the third step

(d) Final distribution of messages

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

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

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

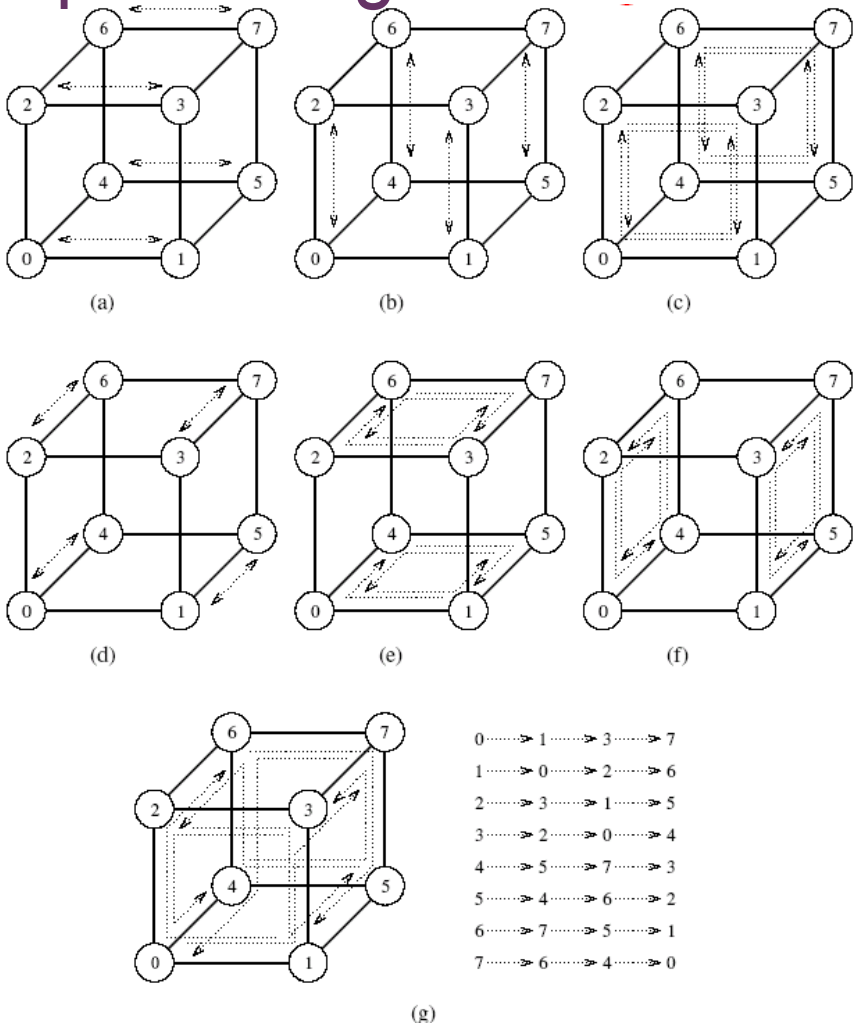
$$T = \sum_{i=1}^{\log p} \left(t_s + t_w m p / 2\right) = \left(t_s + t_w m p / 2\right) \log p$$

- Note!!!: The bisection width of the hypercube is *p/2* so we would expect to be able to communicate the $mp^2/2$ messages between bisections in *O(mp)* time. The above algorithm, with an asymptotic time of *O(mp*log *p)* 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.

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

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

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

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

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

$$T_{comm} = \left(t_s + t_w m\right)\left(p - 1\right)$$

- This is asymptotically optimal in message size.
  - Although asymptotically optimal in message size, this algorithm has a larger growth of the t_s term and so the non-optimal algorithm may still be faster for small messages where the t_s term dominates.
  - In practice, both algorithms are hybridized and the fastest algorithm is selected based on messages size and number of processors

Optimizations of standard algorithms

- Consider the one-to-all broadcast algorithm:
  - Communication time is $(t_s+mt_w)\log p$
  - If the message size is large, then the tree based broadcast will idle processors during the early stages of the algorithm while the large message is transmitted to a few processors (e.g. does the mt_w term need to grow proportional to $\log p$?
  - Is it possible to break up the message into smaller pieces in order to improve processor utilization?
- If the message size is large enough to break into p parts, then we can implement the one-to-all broadcast as a scatter operation to distribute the large message over all processors, then an all-to-all communication can be used to gather the distributed message to all processors. Does this result in a faster communication time?

Optimized one-to-all

- Time for a scatter operation on a hypercube is

$$T_{scatter} = t_s \log p + t_w m(p-1)$$

- Time for the all-to-all operation on a hypercube is

$$T_{all-to-all} = t_s \log p + t_w m(p-1)$$

- Time for scatter then all-to-all with message size of m/p is:

$$T_{one-to-all} = 2\left[t_s \log p + t_w \frac{m}{p}(p-1)\right] \approx 2(t_s \log p + t_w m)$$

Improving Performance of Operations Application of concepts to reductions

- All-to-one reduction can be performed by performing all-to-all reduction (dual of all-to-all broadcast) followed by a gather operation (dual of scatter).
- Since an all-reduce operation is semantically equivalent to an all-to-one reduction followed by a one-to-all broadcast, the asymptotically optimal algorithms for these two operations can be used to construct a similar algorithm for the all-reduce operation.
  - The intervening gather and scatter operations cancel each other. Therefore, an all-reduce operation requires an all-to-all reduction and an all-to-all broadcast.

Discussion