



Introduction to Parallel & Distributed Computing

Programming with MPI Matrix-Vector Multiplication

Lecture 9, Spring 2014

Instructor: 罗国杰

gluo@pku.edu.cn

Overview



- ◆ In the last lecture ...
 - Basics of Message-Passing Interface (MPI)

- ◆ In this lecture ...
 - Example: matrix-vector multiplication
 - Analysis of scalability

Outline: Matrix-Vector Multiplication

- ◆ Sequential algorithm
- ◆ Design, analysis, and implementation of three parallel programs
 - Rowwise block striped
 - Columnwise block striped
 - Checkerboard block

Sequential Algorithm

$$\begin{array}{c} \begin{array}{|c|c|c|c|} \hline 2 & 1 & 0 & 4 \\ \hline 3 & 2 & 1 & 1 \\ \hline 4 & 3 & 1 & 2 \\ \hline 3 & 0 & 2 & 0 \\ \hline \end{array} \times \begin{array}{|c|} \hline 1 \\ \hline 3 \\ \hline 4 \\ \hline 1 \\ \hline \end{array} = \begin{array}{|c|} \hline 9 \\ \hline 14 \\ \hline 19 \\ \hline 11 \\ \hline \end{array} \end{array}$$

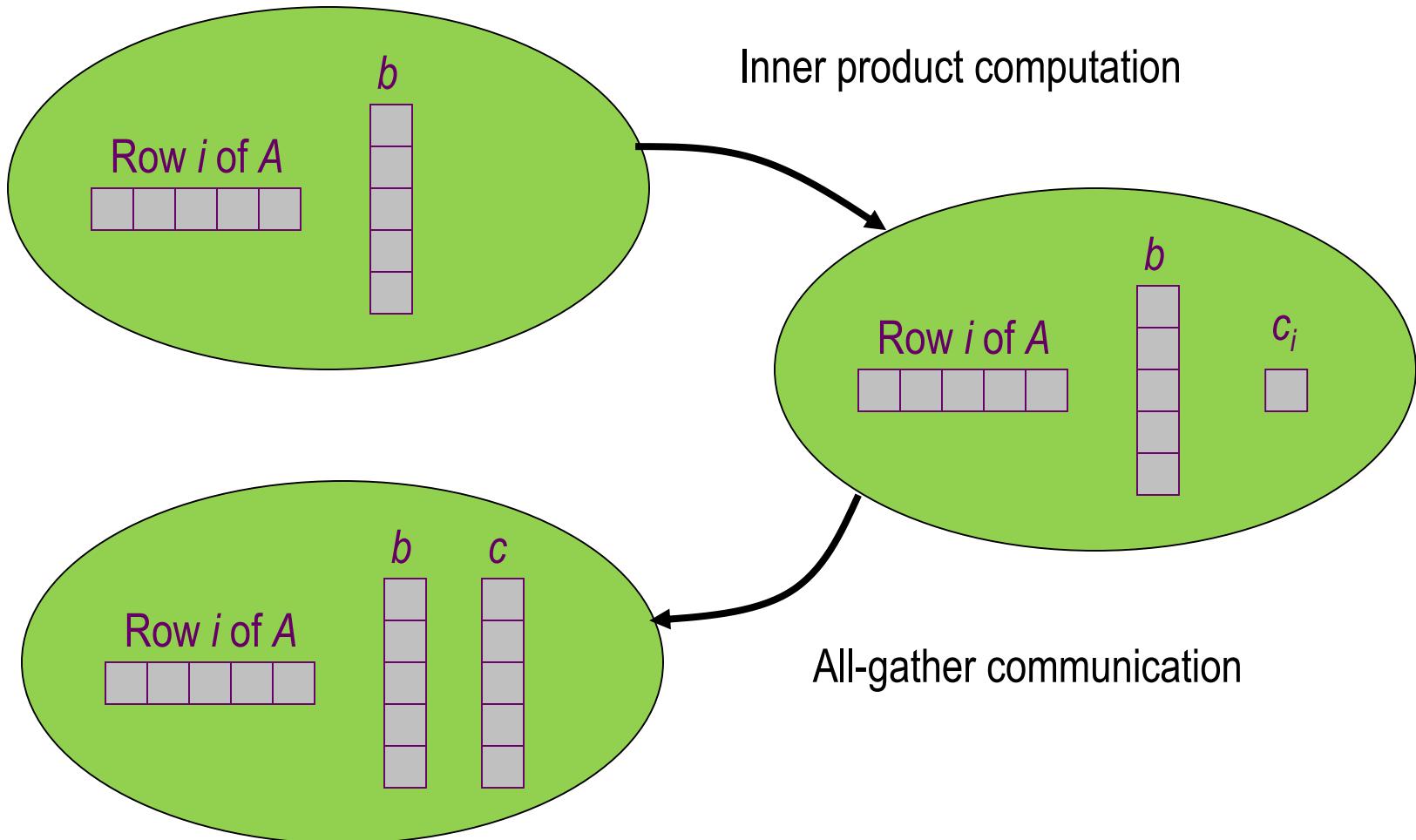
Storing Vectors

- ◆ Divide vector elements among processes, or
- ◆ Replicate vector elements
 - Vector replication acceptable because vectors have only n elements, versus n^2 elements in matrices

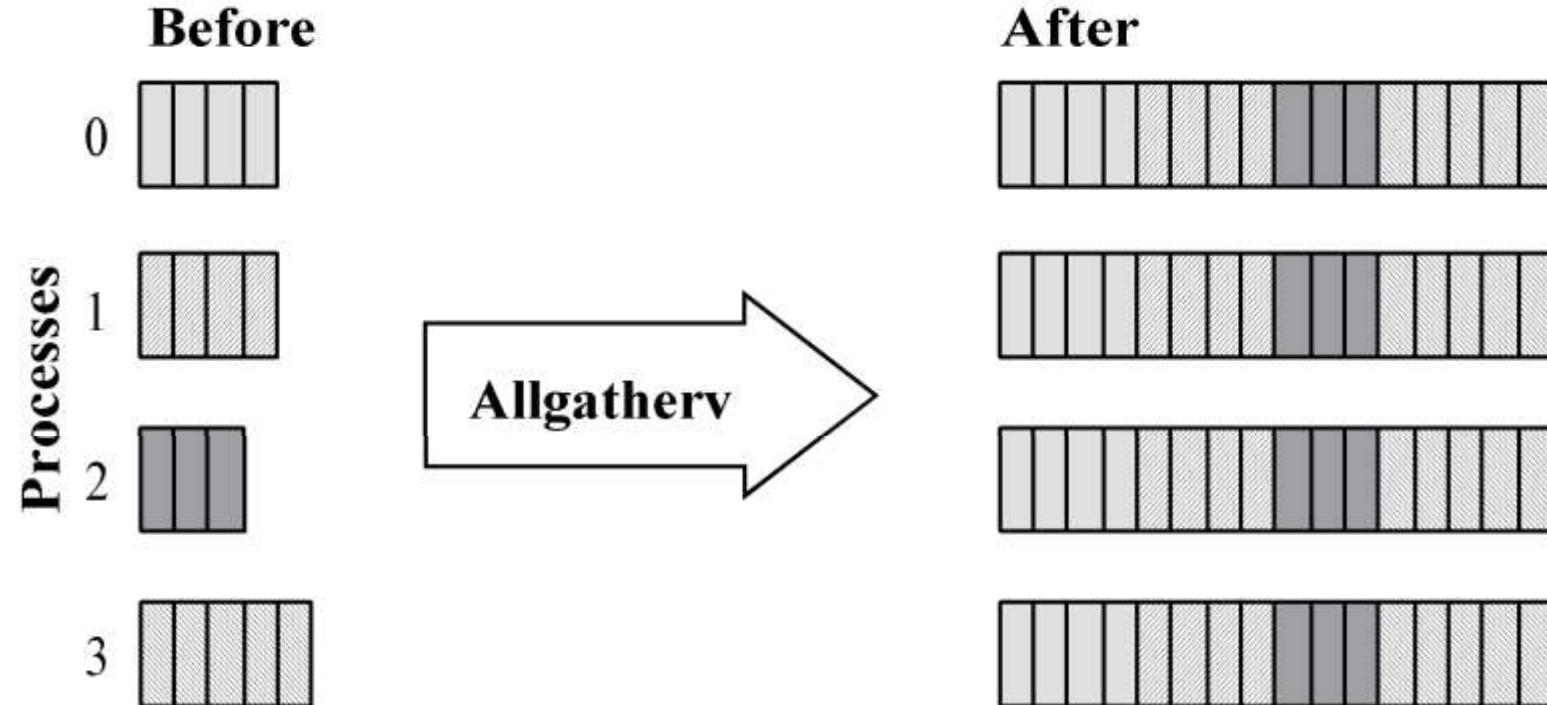
Rowwise Block Striped Matrix

- ◆ Partitioning through domain decomposition
- ◆ Primitive task associated with
 - Row of matrix
 - Entire vector

Phases of Parallel Algorithm



MPI_Allgatherv

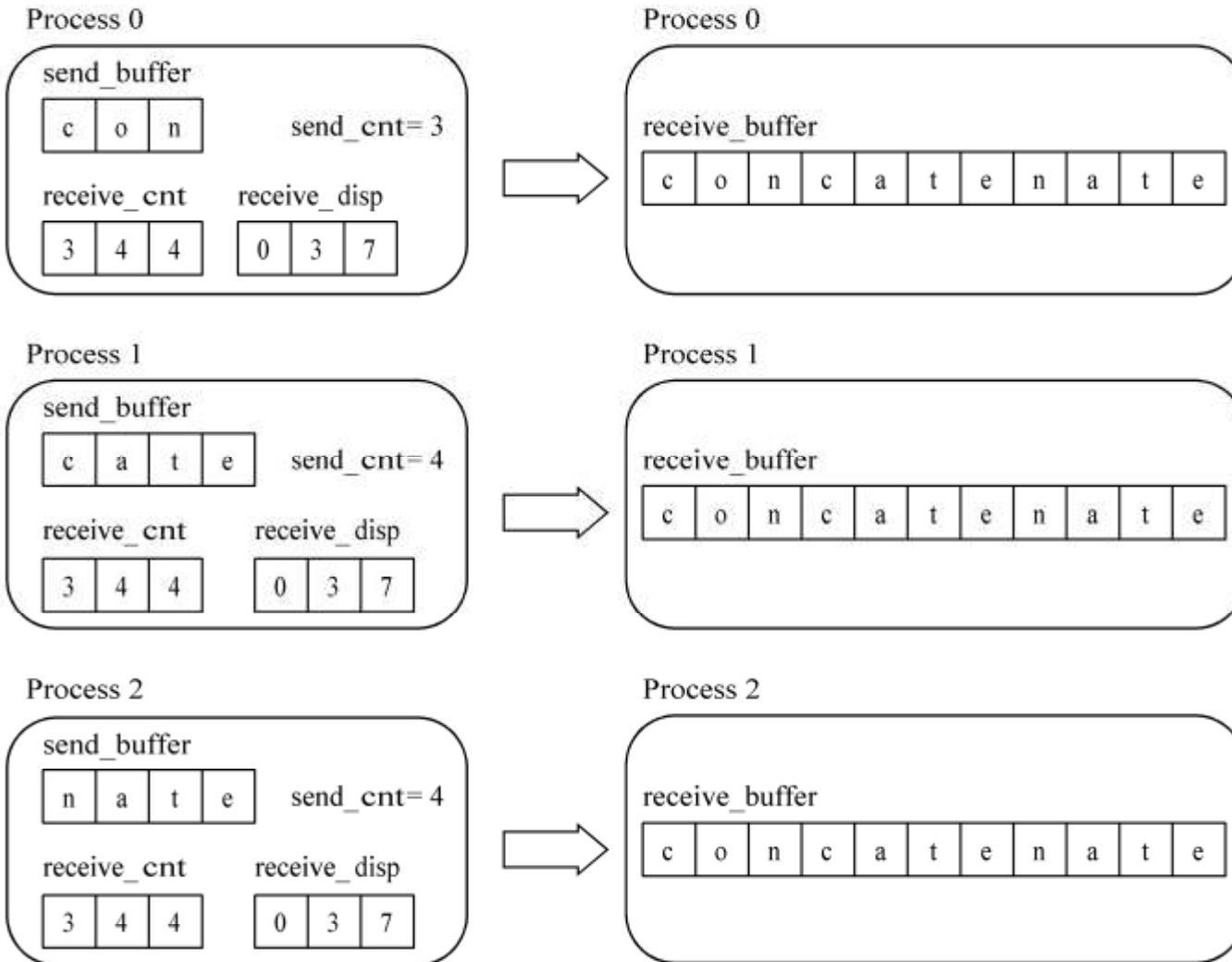


MPI_Allgatherv

```
int MPI_Allgatherv (
    void *send_buffer,
    int send_cnt,
    MPI_Datatype send_type,
    void *receive_buffer,
    int *receive_cnt,
    int *receive_disp,
    MPI_Datatype receive_type,
    MPI_Comm communicator)
```

**Use a simpler function *MPI_Allgather(...)*
when the number of elements per processor
is a constant**

MPI_Allgatherv in Action



Agglomeration and Mapping

- ◆ **Static number of tasks**
- ◆ **Regular communication pattern (all-gather)**
- ◆ **Computation time per task is constant**
- ◆ **Strategy:**
 - **Agglomerate groups of rows**
 - **Create one task per MPI process**

Columnwise Block Striped Matrix

- ◆ Partitioning through domain decomposition
- ◆ Task associated with
 - Column of matrix
 - Vector element

Matrix-Vector Multiplication

$$\begin{aligned}c_0 &= a_{0,0} b_0 + a_{0,1} b_1 + a_{0,2} b_2 + a_{0,3} b_3 + a_{4,4} b_4 \\c_1 &= a_{1,0} b_0 + a_{1,1} b_1 + a_{1,2} b_2 + a_{1,3} b_3 + a_{1,4} b_4 \\c_2 &= a_{2,0} b_0 + a_{2,1} b_1 + a_{2,2} b_2 + a_{2,3} b_3 + a_{2,4} b_4 \\c_3 &= a_{3,0} b_0 + a_{3,1} b_1 + a_{3,2} b_2 + a_{3,3} b_3 + b_{3,4} b_4 \\c_4 &= a_{4,0} b_0 + a_{4,1} b_1 + a_{4,2} b_2 + a_{4,3} b_3 + \underbrace{a_{4,4} b_4}_{\text{Proc 4}}\end{aligned}$$

Processor 1's initial computation

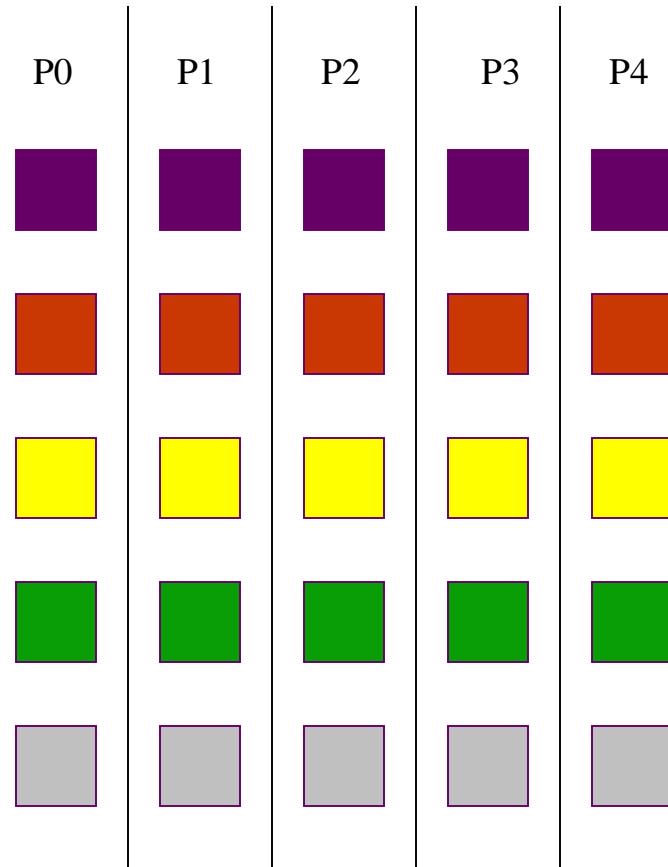
Processor 0's initial computation

Proc 3

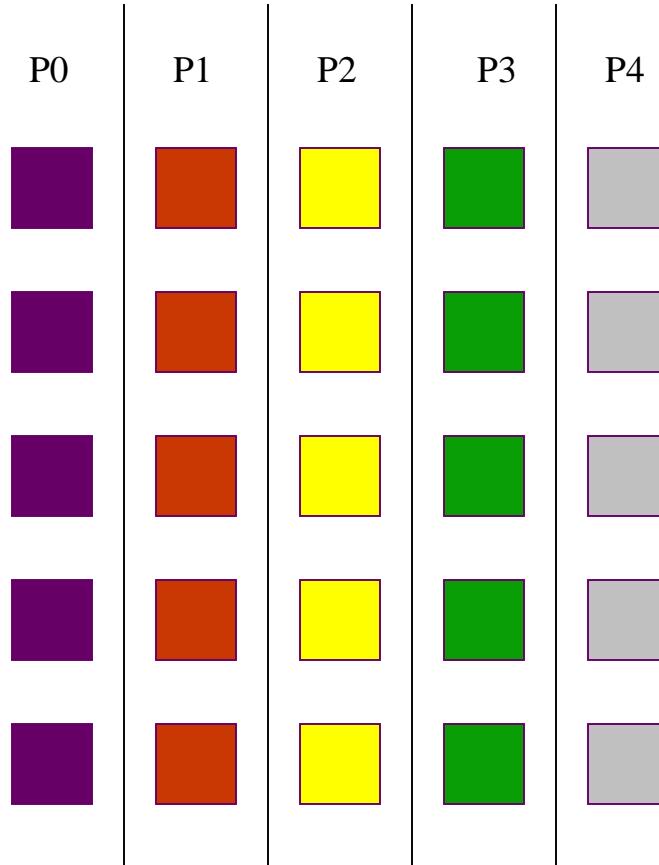
Proc 2

Proc 4

All-to-all Exchange (Before)



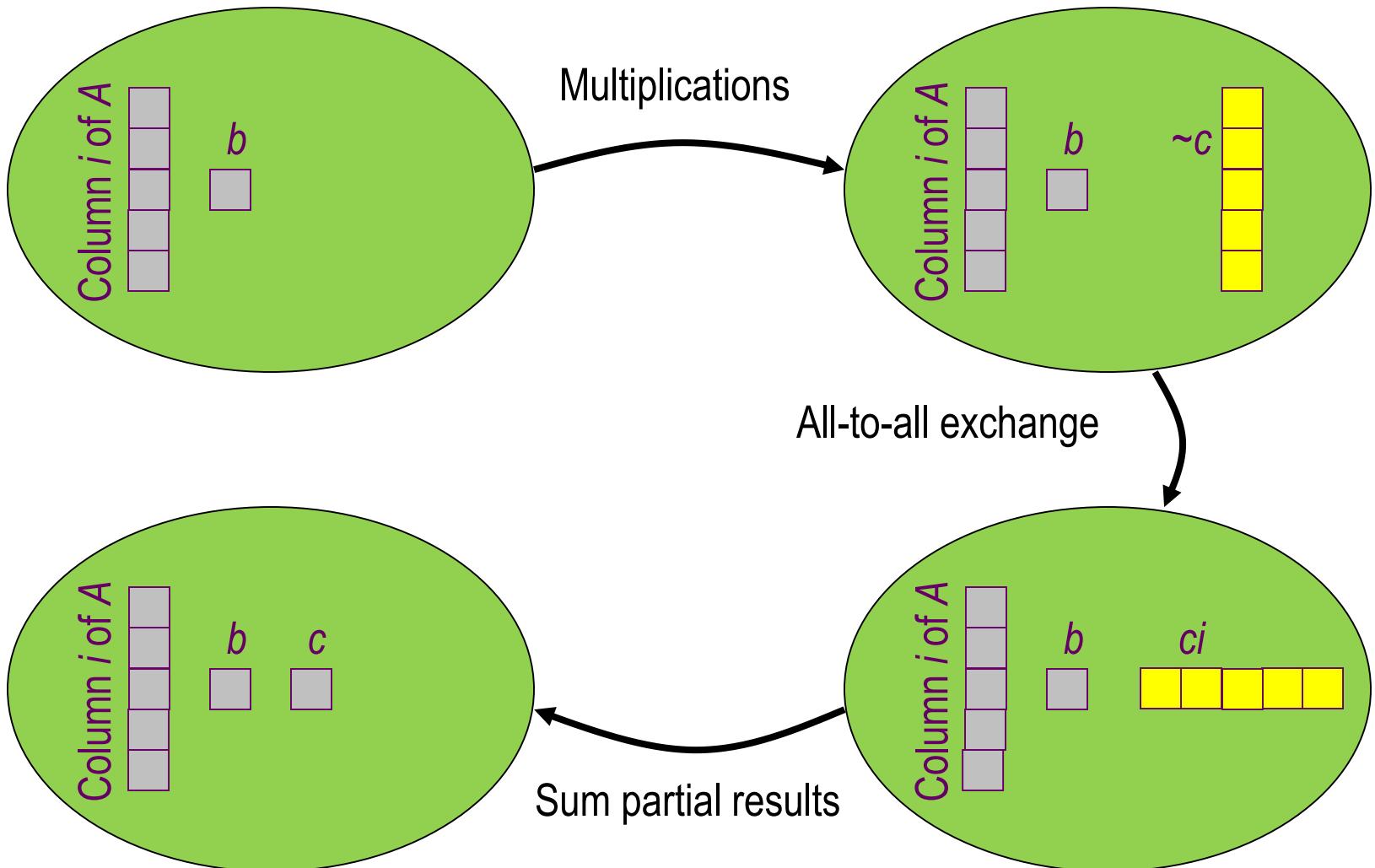
All-to-all Exchange (After)



All-to-All Exchange (All-to-All Personalized Communication)



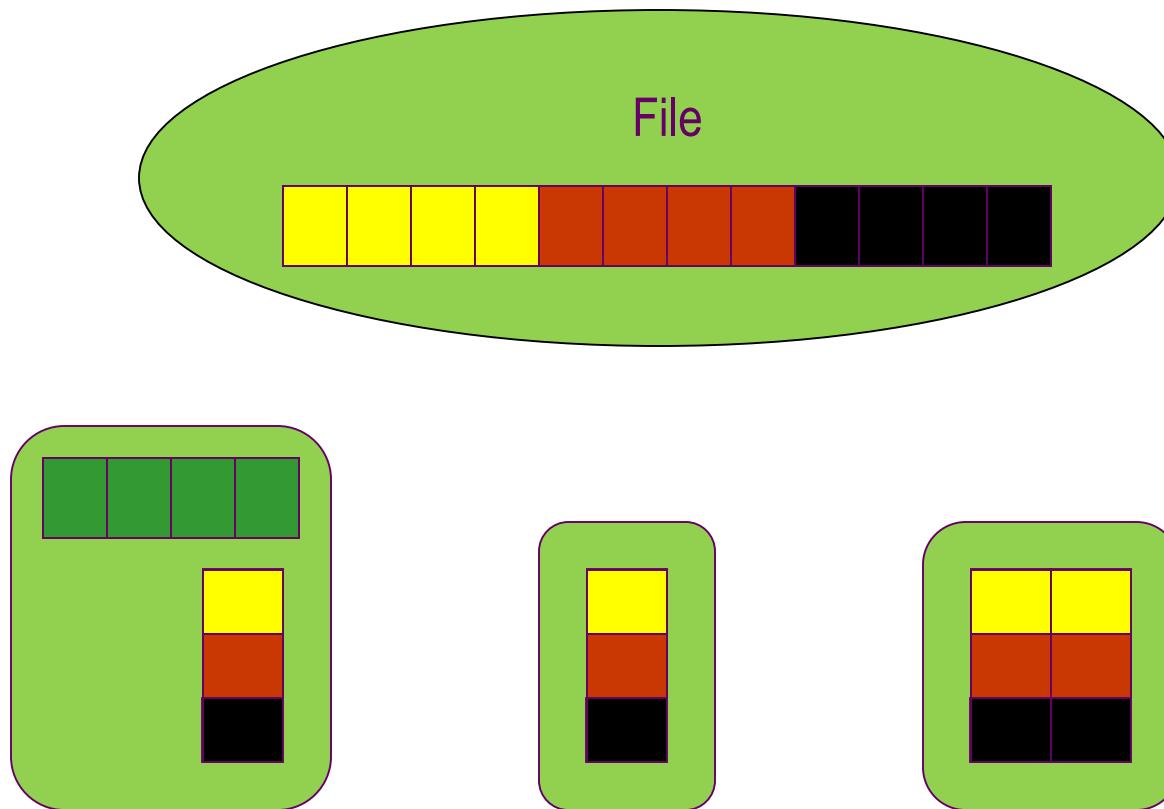
Phases of Parallel Algorithm



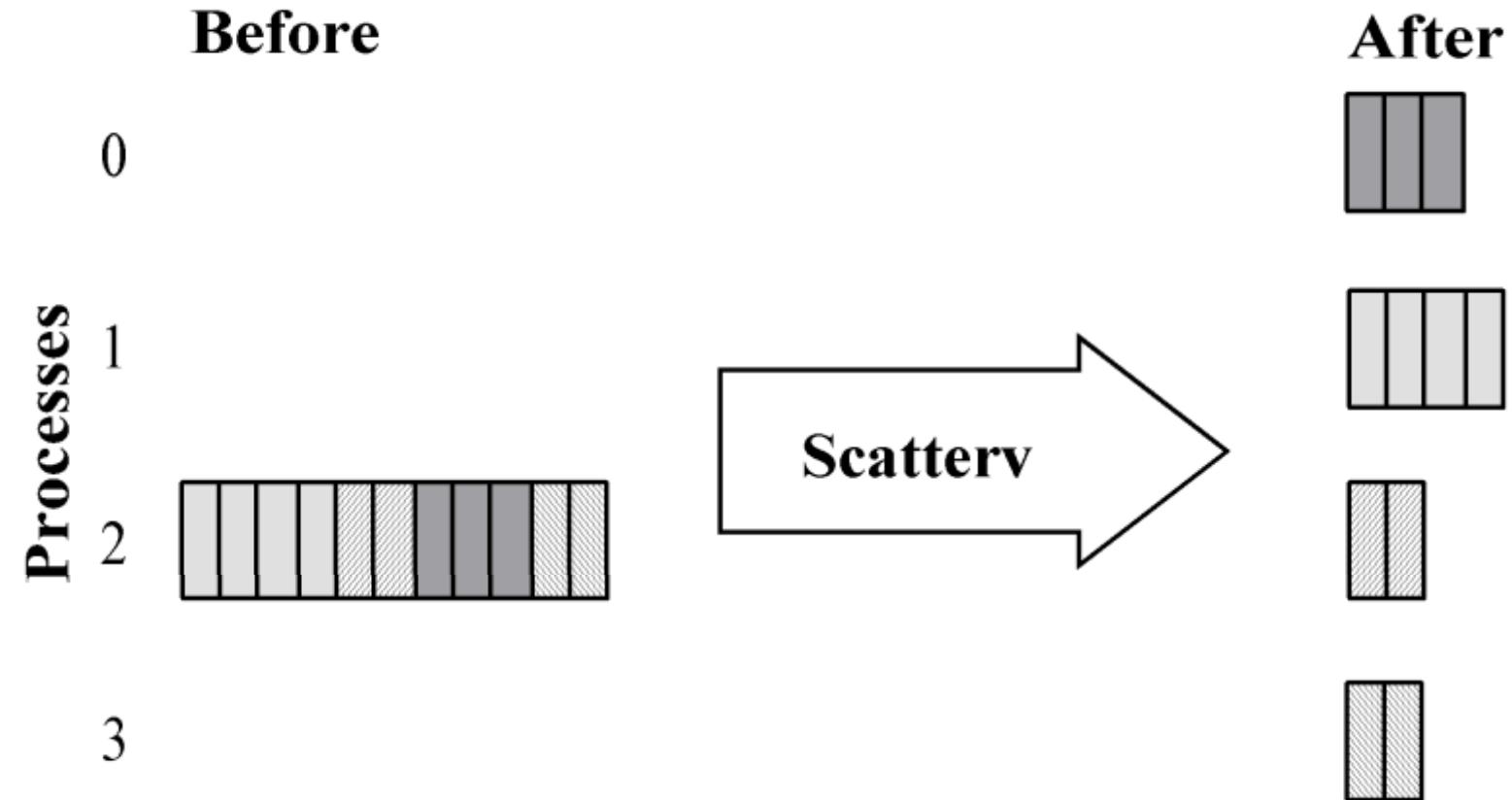
Agglomeration and Mapping

- ◆ **Static number of tasks**
- ◆ **Regular communication pattern (all-to-all)**
- ◆ **Computation time per task is constant**
- ◆ **Strategy:**
 - **Agglomerate groups of columns**
 - **Create one task per MPI process**

Reading a Block-Column Matrix



MPI_Scatterv



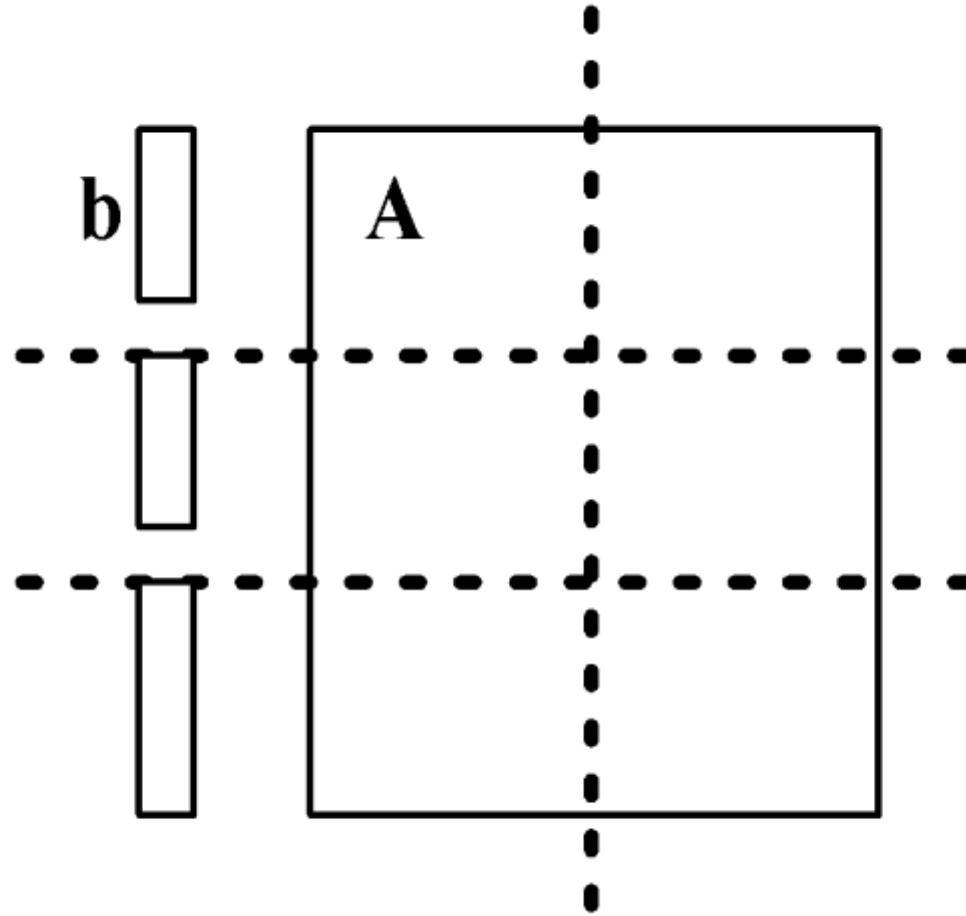
Header for MPI_Scatterv

```
int MPI_Scatterv (  
    void *send_buffer,  
    int *send_cnt,  
    int *send_disp,  
    MPI_Datatype send_type,  
    void *receive_buffer,  
    int receive_cnt,  
    MPI_Datatype receive_type,  
    int root,  
    MPI_Comm communicator)
```

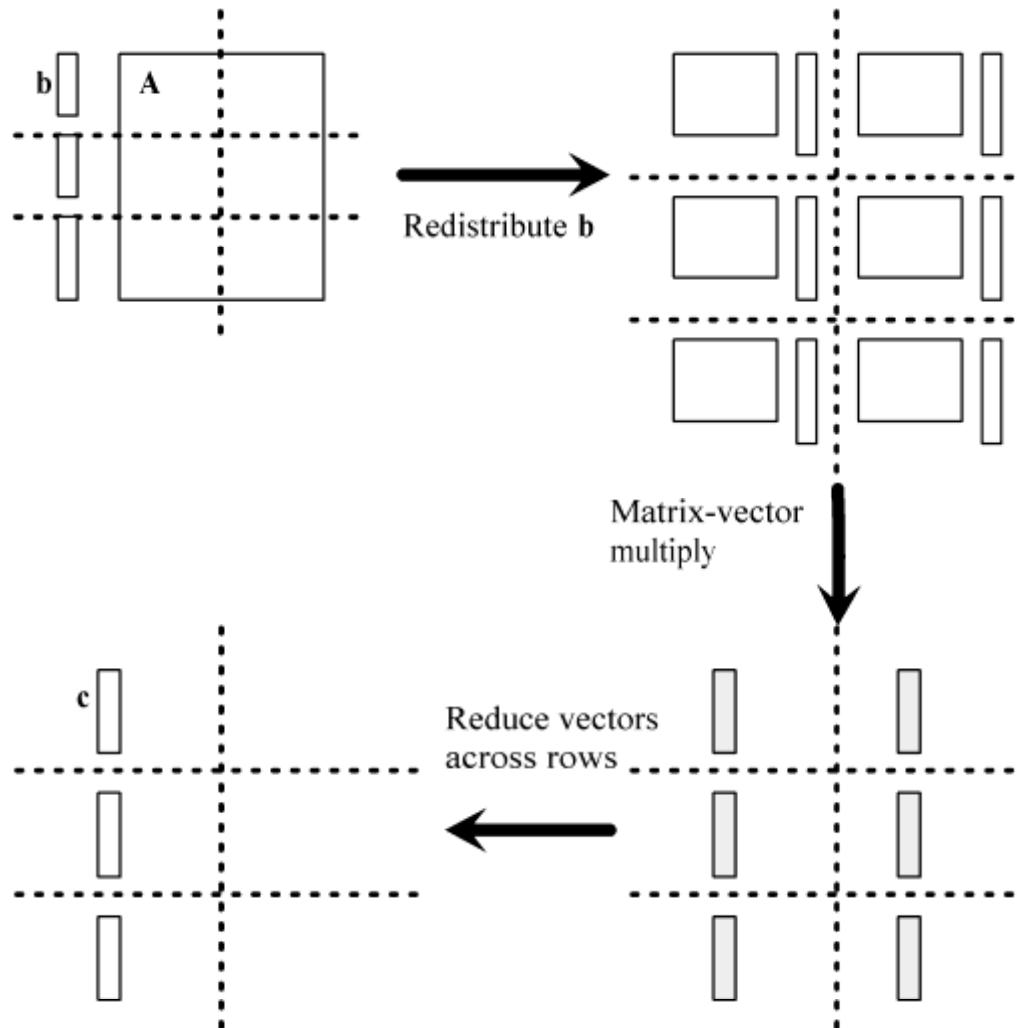
Checkerboard Block Decomposition

- ◆ Associate primitive task with each element of the matrix A
- ◆ Each primitive task performs one multiply
- ◆ Agglomerate primitive tasks into rectangular blocks
- ◆ Processes form a 2-D grid
- ◆ Vector b distributed by blocks among processes in first column of grid

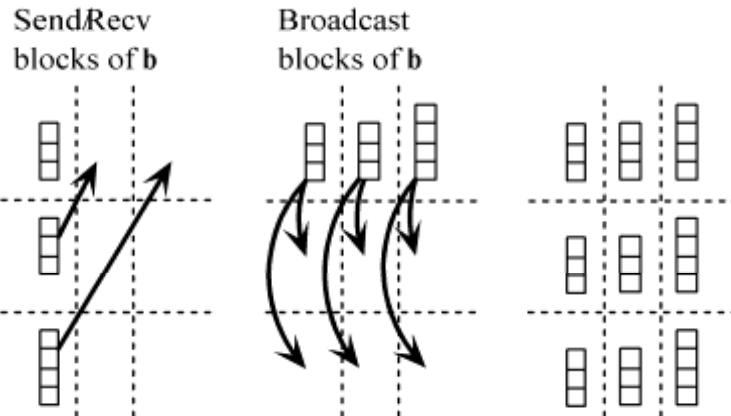
Tasks after Agglomeration



Algorithm's Phases

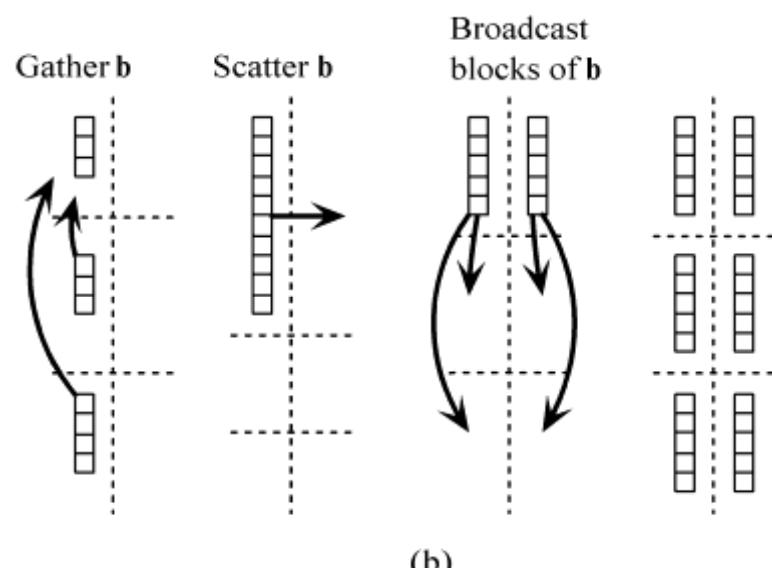


Redisistributing Vector b



(a)

When p is a
square
number



(b)

Redistributing Vector b

- ◆ Step 1: Move b from processes in first row to processes in first column
 - If p square
 - First column/first row processes send/receive portions of b
 - If p not square
 - Gather b on process 0, 0
 - Process 0, 0 broadcasts to first row procs
- ◆ Step 2: First row processes scatter b within columns

Creating Communicators

- ◆ Want processes in a virtual 2-D grid
- ◆ Create a custom communicator to do this
- ◆ Collective communications involve all processes in a communicator
- ◆ We need to do broadcasts, reductions among subsets of processes
- ◆ We will create communicators for processes in same row or same column

What's In a Communicator?

◆ **Process group**

◆ **Context**

◆ **Attributes**

- **Topology** (lets us address processes another way)
- **Others we won't consider**

Creating 2-D Virtual Grid of Processes

◆ *MPI_Dims_create*

- **Input parameters**
 - Total number of processes in desired grid
 - Number of grid dimensions
- **Returns number of processes in each dim**

◆ *MPI_Cart_create*

- **Creates communicator with Cartesian topology**

MPI_Dims_create

```
int MPI_Dims_create (
    int nodes,
        /* Input - Procs in grid */
    int dims,
        /* Input - Number of dims */
    int *size)
        /* Input/Output - Size of
           each grid dimension */
```

dims before call	function call	dims on return
(0,0)	MPI_DIMS_CREATE(6, 2, dims)	(3,2)
(0,0)	MPI_DIMS_CREATE(7, 2, dims)	(7,1)
(0,3,0)	MPI_DIMS_CREATE(6, 3, dims)	(2,3,1)
(0,3,0)	MPI_DIMS_CREATE(7, 3, dims)	erroneous call

source: DeinoMPI

Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.

MPI_Cart_create

```
int MPI_Cart_create (
    MPI_Comm old_comm, /* Input - old communicator */
    int dims, /* Input - grid dimensions */
    int *size, /* Input - # procs in each dim */
    int *periodic,
        /* Input - periodic[j] is 1 if dimension j
           wraps around; 0 otherwise */
    int reorder,
        /* 1 if process ranks can be reordered */
    MPI_Comm *cart_comm)
    /* Output - new communicator */
```

Using MPI_Dims_create and MPI_Cart_create

```
MPI_Comm cart_comm;
int p;
int periodic[2];
int size[2];

...
size[0] = size[1] = 0;
MPI_Dims_create (p, 2, size);
periodic[0] = periodic[1] = 0;
MPI_Cart_create (MPI_COMM_WORLD, 2, size,
1, &cart_comm);
```

Useful Grid-related Functions

- ◆ ***MPI_Cart_rank***
 - Given coordinates of process in Cartesian communicator, returns process rank

- ◆ ***MPI_Cart_coords***
 - Given rank of process in Cartesian communicator, returns process' coordinates

Header for MPI_Cart_rank

```
int MPI_Cart_rank (
    MPI_Comm comm,
        /* In - Communicator */
    int *coords,
        /* In - Array containing process'
           grid location */
    int *rank)
        /* Out - Rank of process at
           specified coords */
```

Header for MPI_Cart_coords

```
int MPI_Cart_coords (  
    MPI_Comm comm,  
    /* In - Communicator */  
    int rank,  
    /* In - Rank of process */  
    int dims,  
    /* In - Dimensions in virtual grid  
 * /  
    int *coords)  
    /* Out - Coordinates of specified  
       process in virtual grid */
```

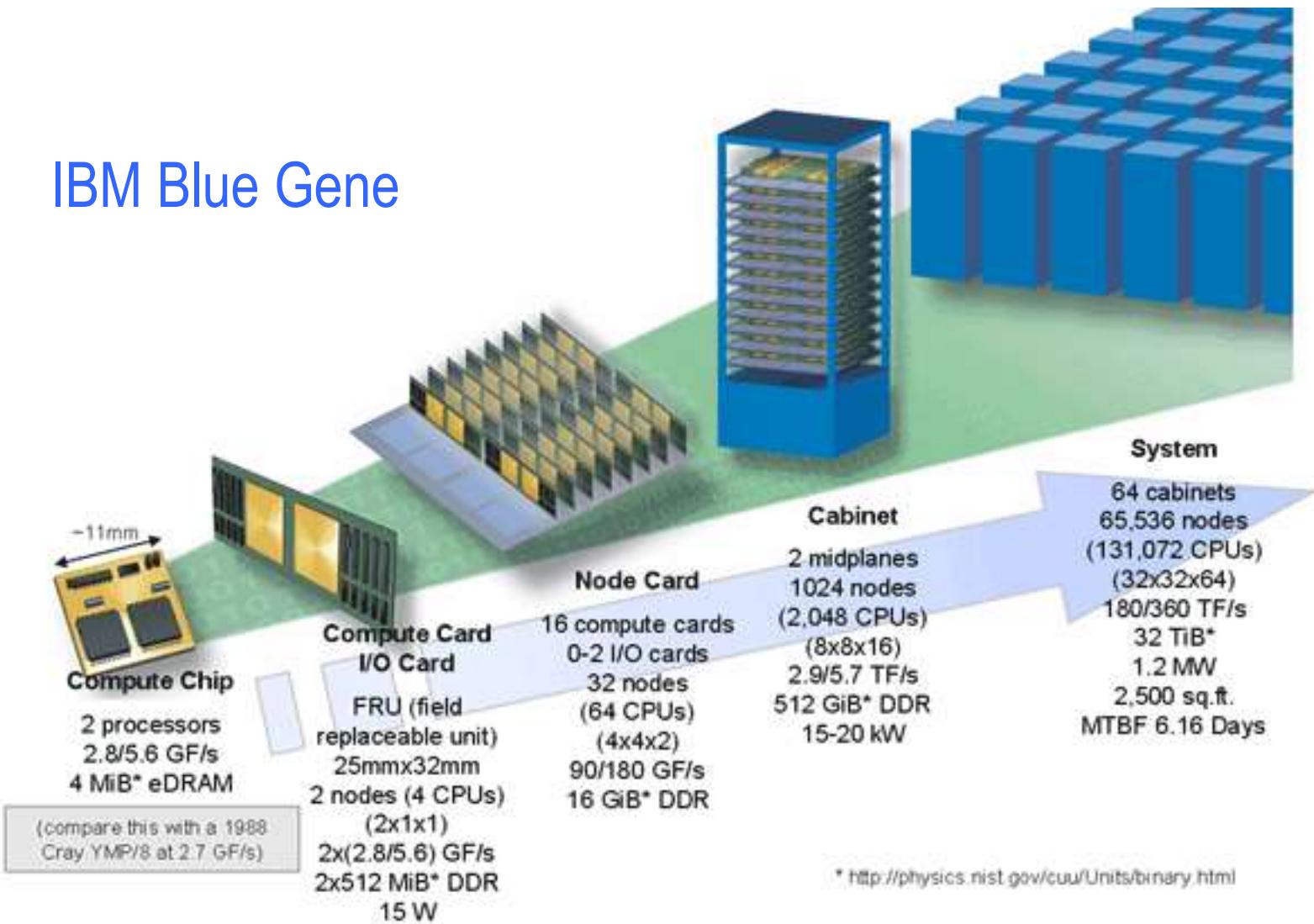
Collective Communications

	Ring	2-D Mesh	Hypercube
One-to-All Broadcast			
<i>All-to-One Reduction</i>			
All-to-All Broadcast			
<i>All-to-All Reduction</i>			
Scatter			
Gather			
All-to-All Personalized Communication			

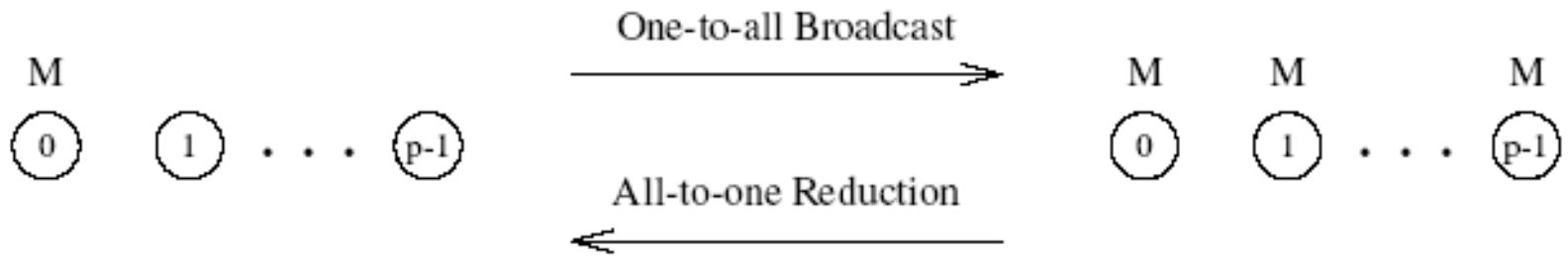
Dual of a communication operation

Multi-Dimensional Structure

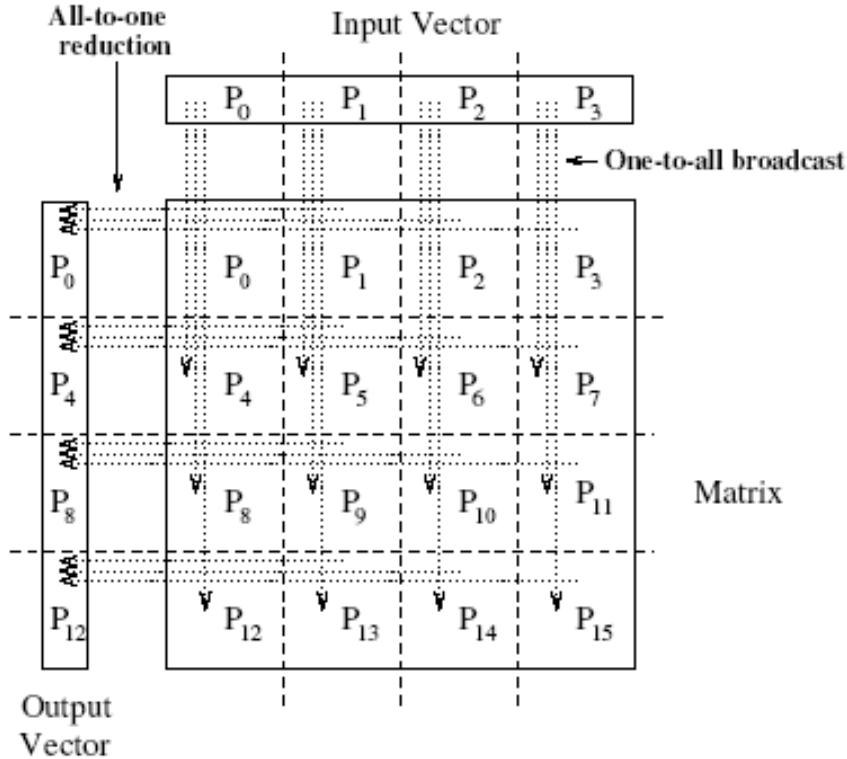
IBM Blue Gene



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



Example

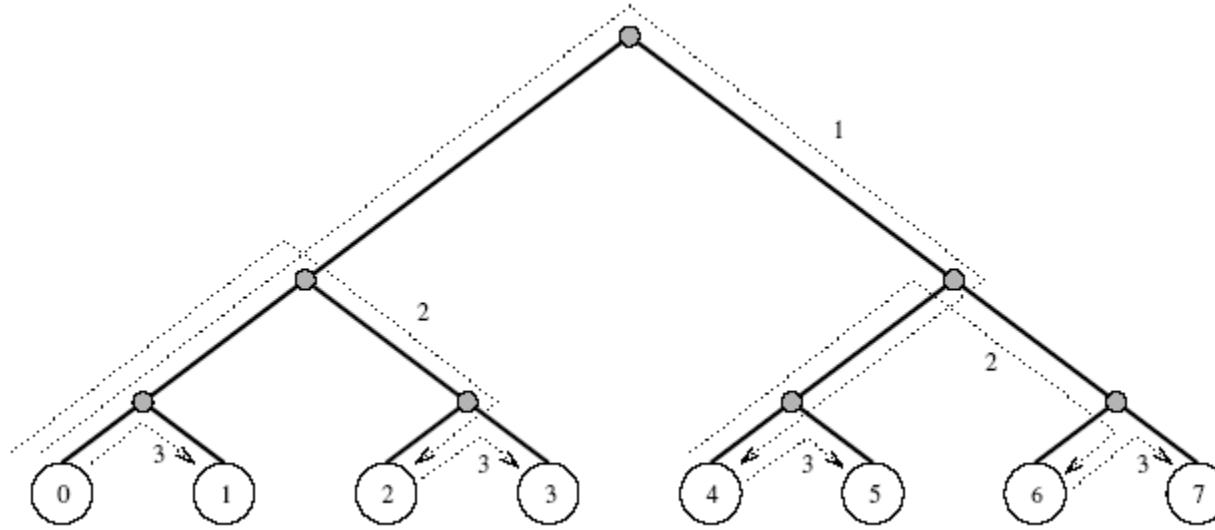


**One-to-all broadcast and all-to-one reduction
in the multiplication of a 4×4 matrix with a 4×1 vector.**

Approaches

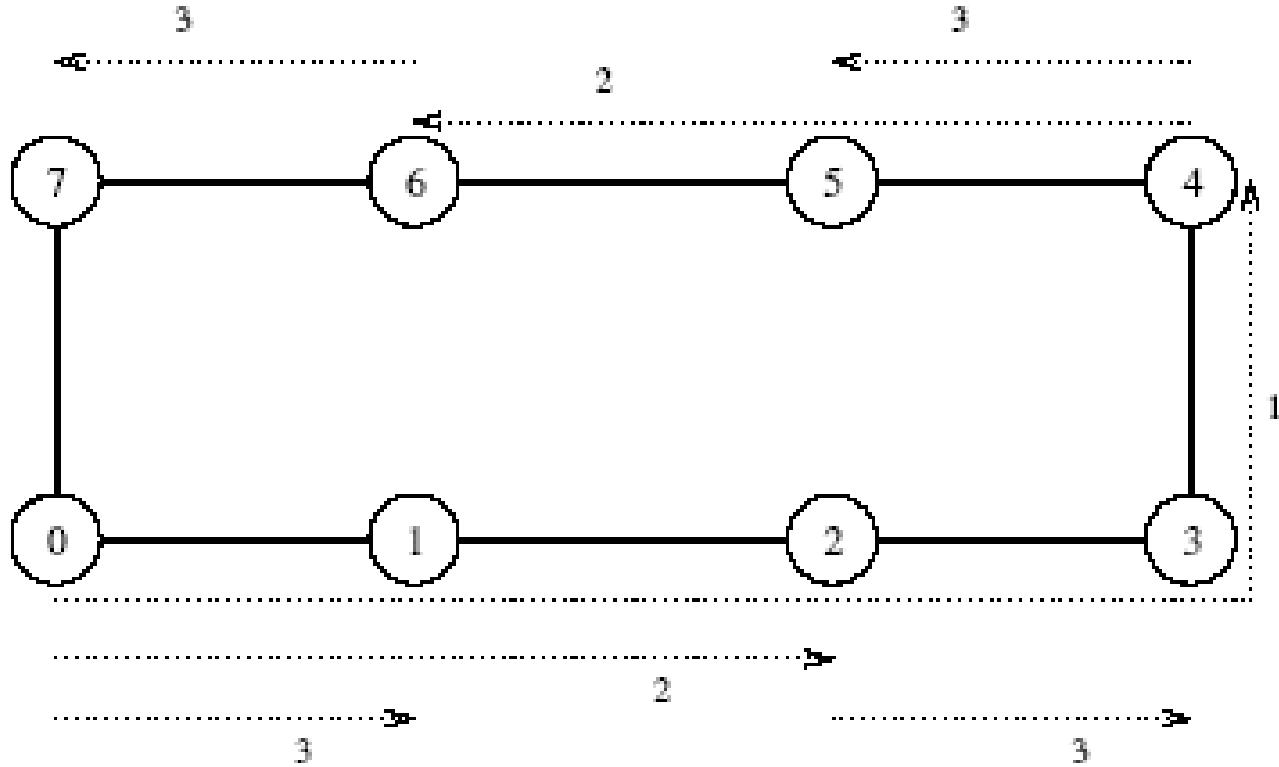
- ◆ Simplest way is to send $p-1$ messages from the source to the other $p-1$ processors - this is not very efficient.
- ◆ Use *recursive doubling*: source sends a message to a selected processor. We now have two independent problems defined over halves of machines.
- ◆ Reduction can be performed in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

Broadcast and Reduction on a Balanced Binary Tree

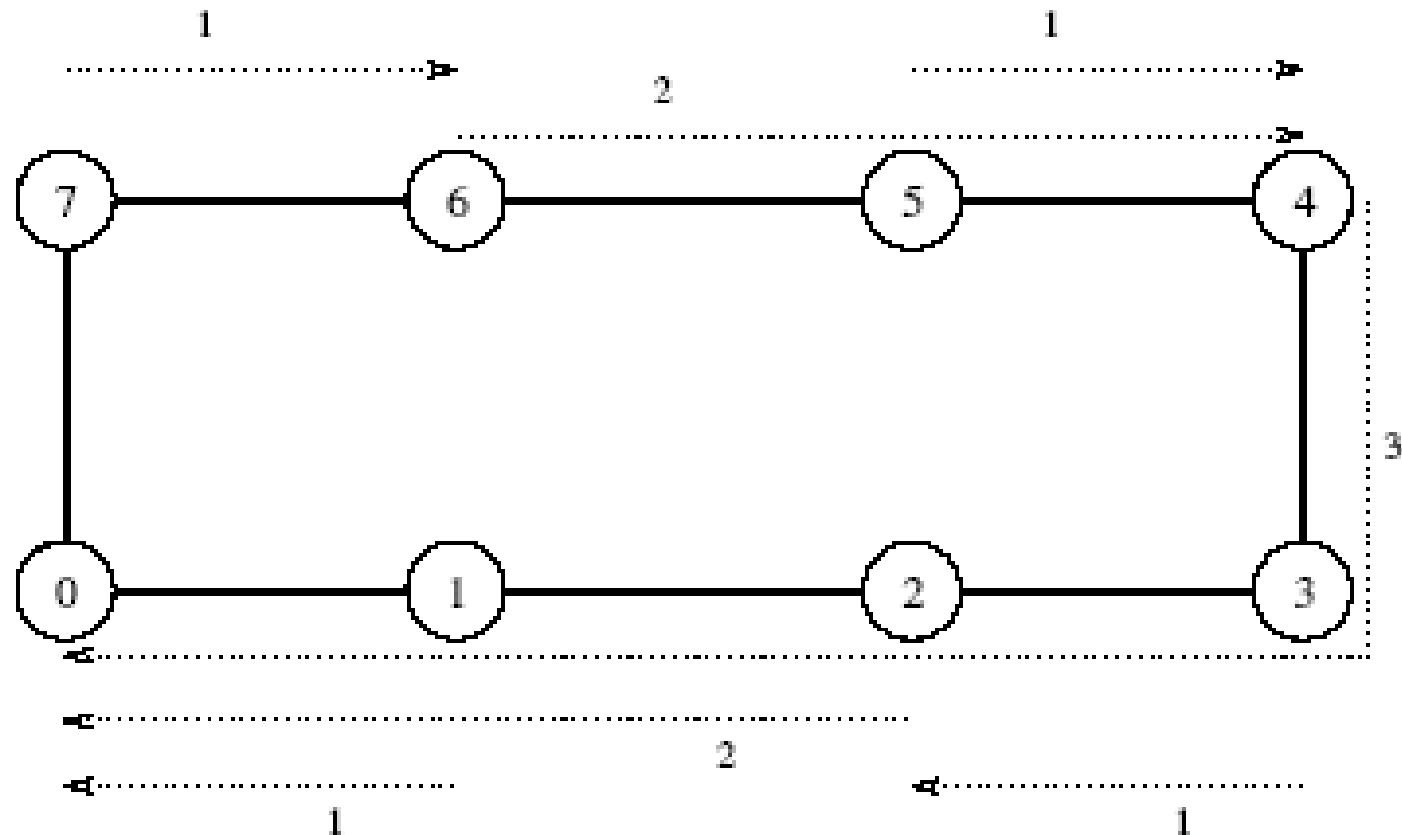


One-to-all broadcast on an eight-node tree.

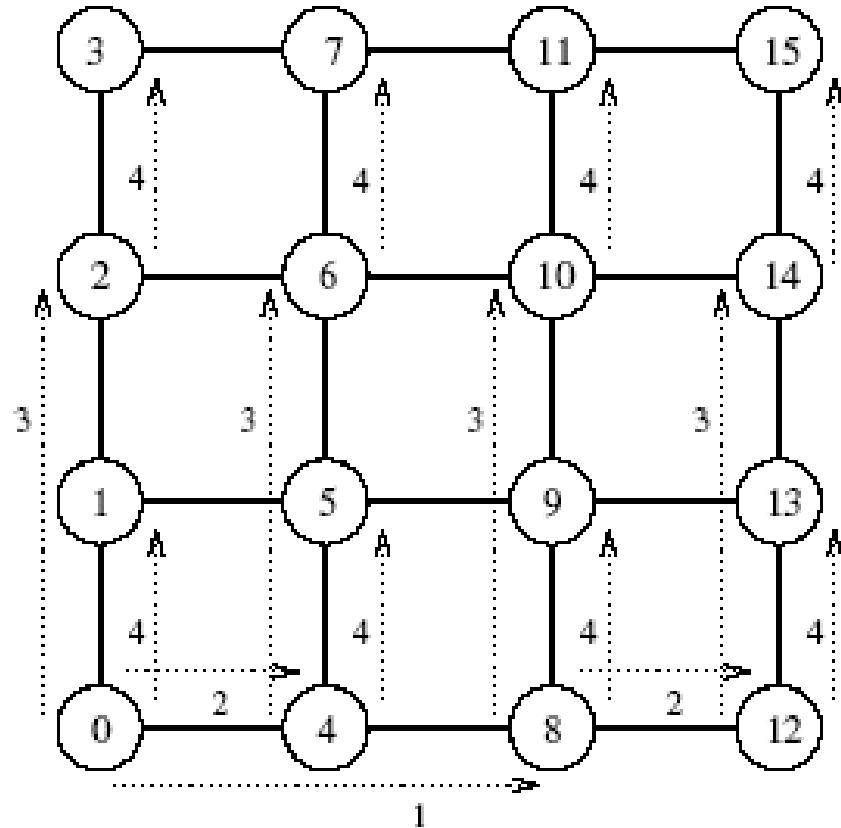
One-to-All Broadcast on Ring



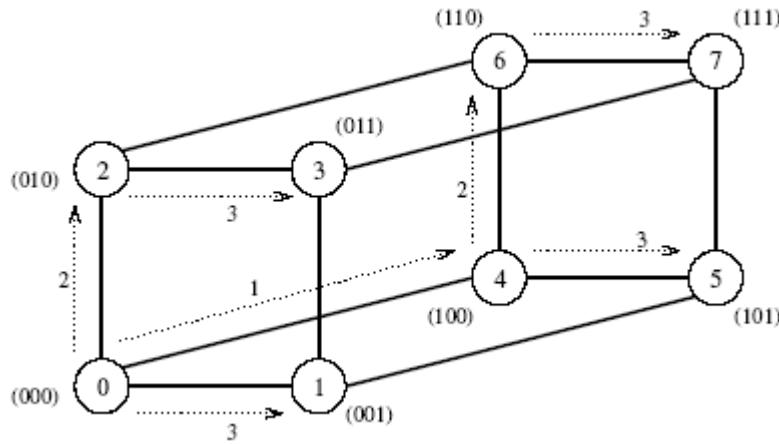
All-to-One Reduction



Broadcast and Reduction on a Mesh: Example



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

```
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);
11.                 /* Convert virtual_dest to the label of the physical destination */
12.                 else
13.                     virtual_source := my_virtual_id XOR  $2^i$ ;
14.                     receive X from (virtual_source XOR source);
15.                     /* Convert virtual_source to the label of the physical source */
16.                     endelse;
17.                 endfor;
18.             end GENERAL ONE TO ALL BC
```

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

Broadcast and Reduction Algorithms

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

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

Point-to-Point Communication Cost in Parallel Systems (1/2)

◆ The principal parameters

- Setup time (t_s), including
 - The time to prepare the message
 - The time to execute the routing algorithm
 - The time to setup an interface between the node and the router
- Per-hop time (t_h)
 - The time to travel between two *directly-connected* nodes
- Per-word transfer time (t_w)
 - Comes from network overheads and buffering overheads

◆ Pass a message of size m between two nodes l hops away

- $t_{comm} = t_s + lt_h + mt_w$

Point-to-Point Communication Cost in Parallel Systems (2/2)

- ◆ A simplified cost model in an uncongested network

- $t_{comm} = t_s + mt_w$

- ◆ Reasons to drop the distance term

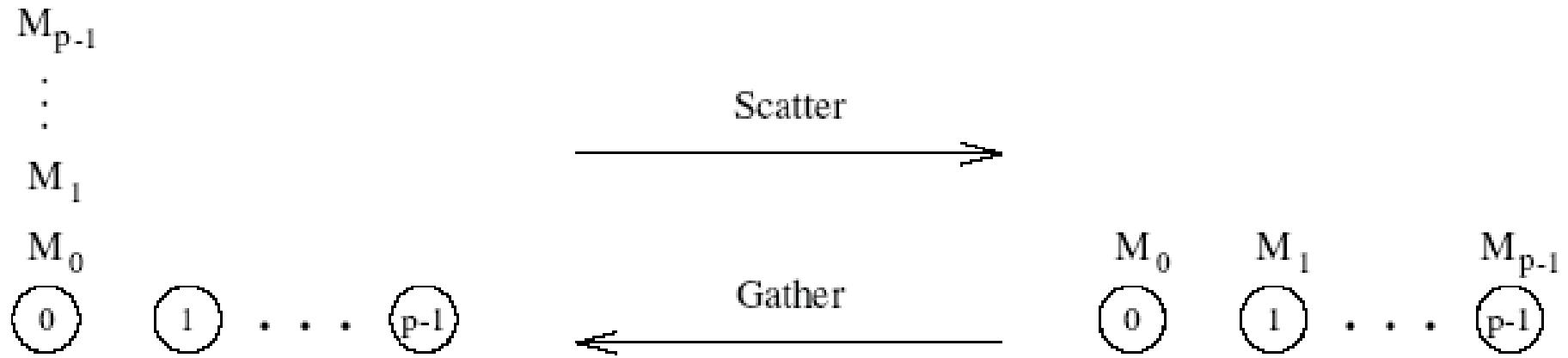
- In many message-passing libraries (e.g., MPI), the programmer has little control on the mapping of processes onto physical processors
- Many architectures rely on randomized routing to alleviate hot-spots and contention on the network. It yields no benefits to minimize the number of hops
- The per-hop time is dominated by either t_s (for small messages) or mt_w (for large messages) in most networks of parallel systems

Cost Analysis of One-to-All Broadcast and All-to-One Reduction

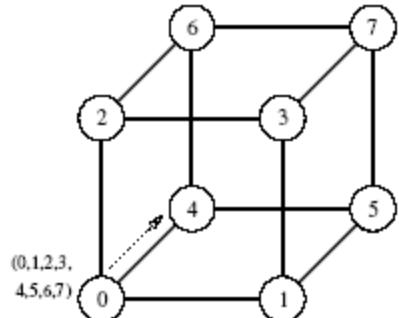
- ◆ The broadcast or reduction procedure involves $\log p$ point-to-point simple message transfers, each at a time cost of $t_s + t_w m$.
- ◆ The total time is therefore given by:

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

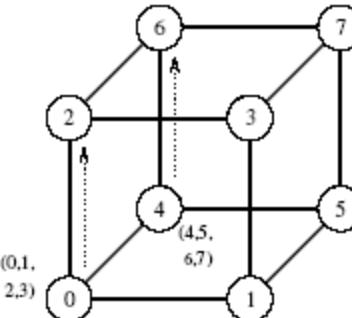
Gather and Scatter Operations



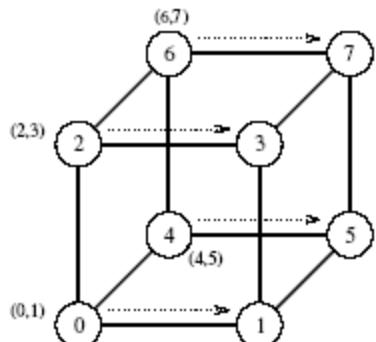
Example of the Scatter Operation



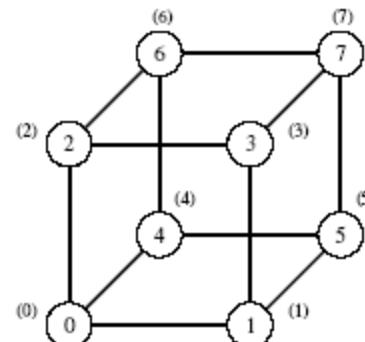
(a) Initial distribution of messages



(b) Distribution before the second step



(c) Distribution before the third step



(d) Final distribution of messages

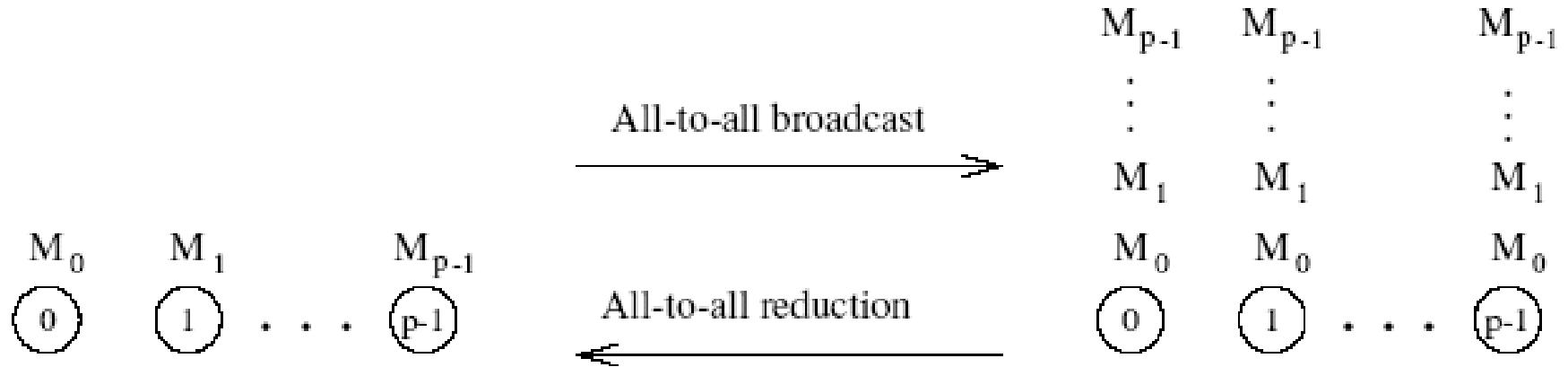
The scatter operation on an eight-node hypercube.

Cost of Scatter and Gather

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

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

All-to-All Broadcast and Reduction



Approaches

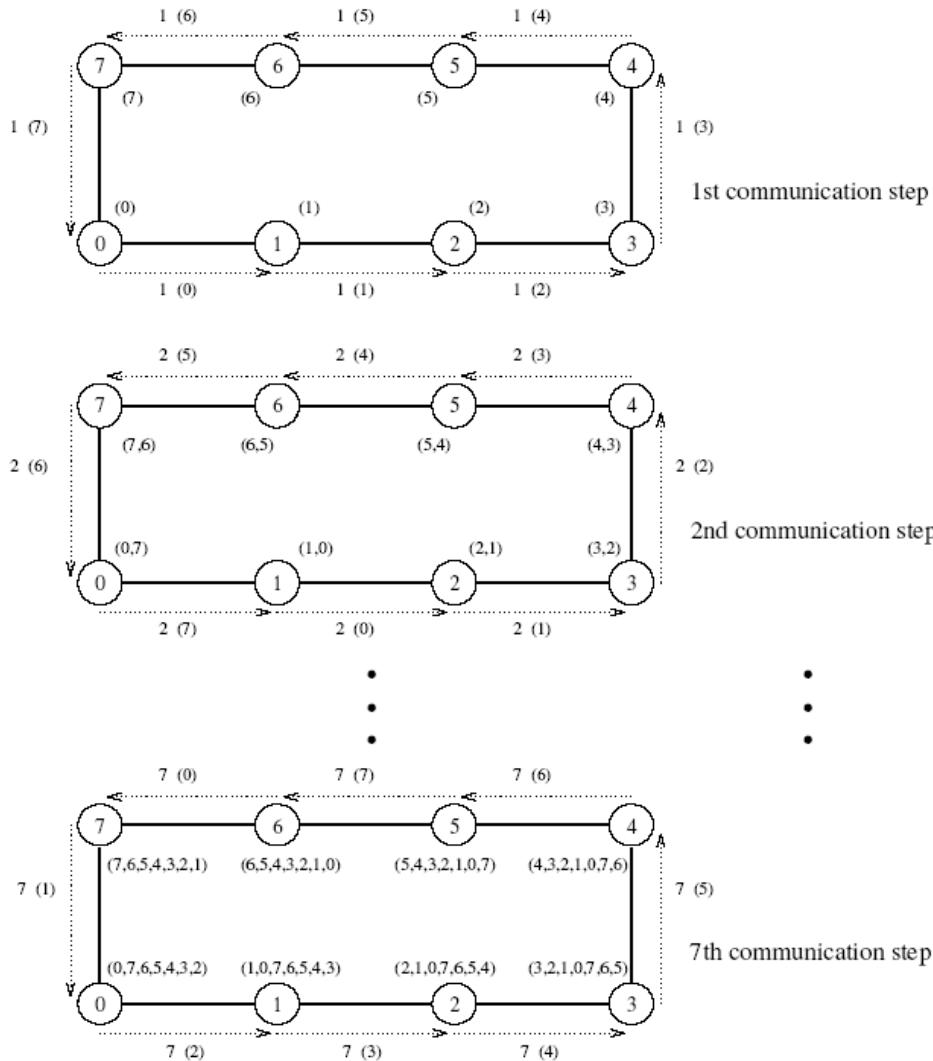
◆ Simplest approach

- Perform p one-to-all broadcasts.
- This is not the most efficient way, though.

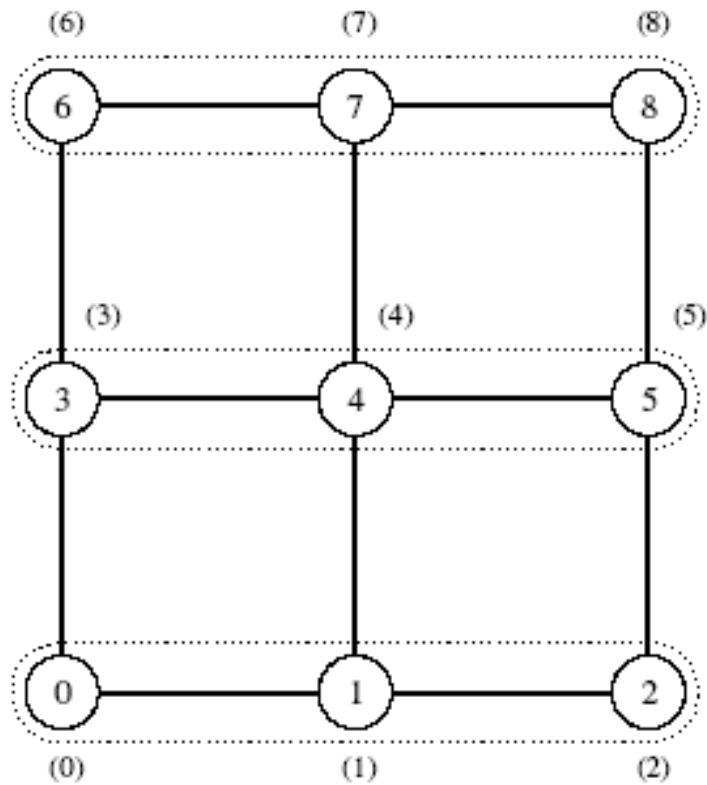
◆ Better approach

- 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.

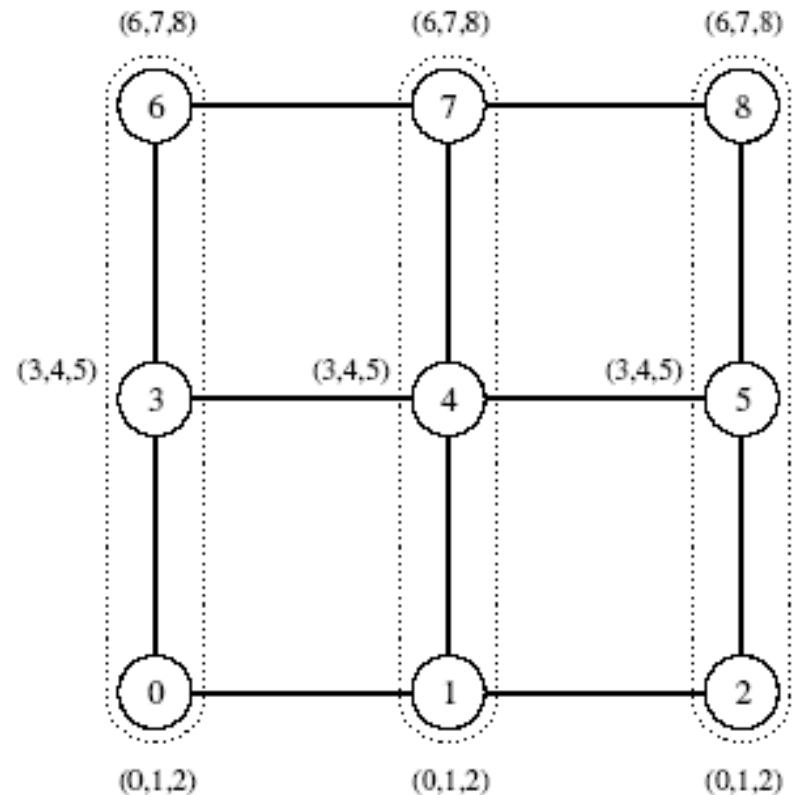
All-to-All Broadcast and Reduction on a Ring



All-to-all Broadcast on a Mesh

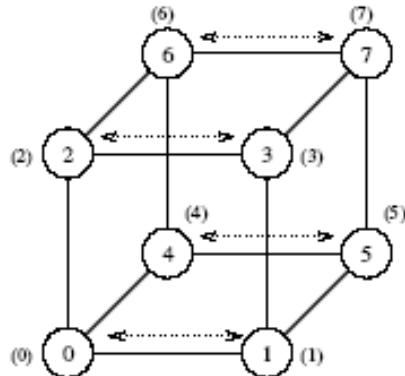


(a) Initial data distribution

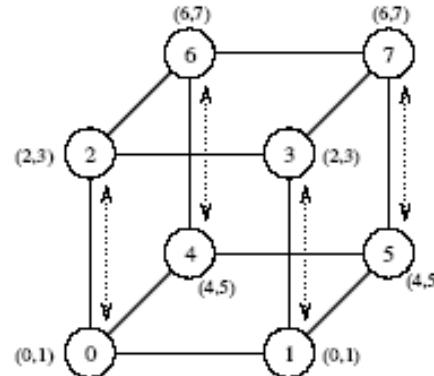


(b) Data distribution after rowwise broadcast

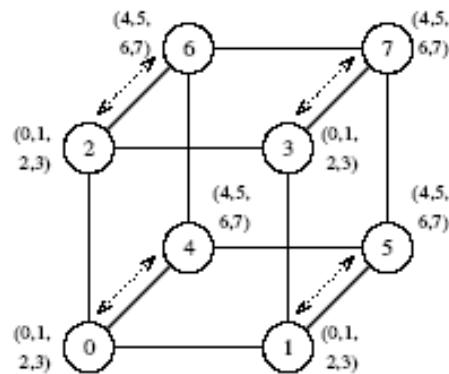
All-to-all broadcast on a Hypercube



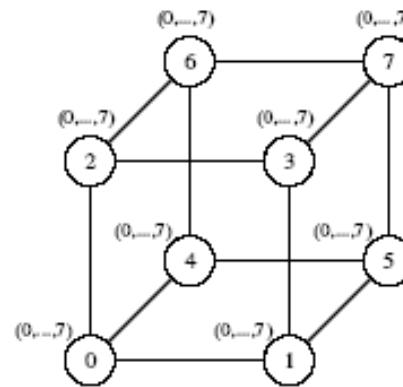
(a) Initial distribution of messages



(b) Distribution before the second step



(c) Distribution before the third step



(d) Final distribution of messages

All-to-all Reduction

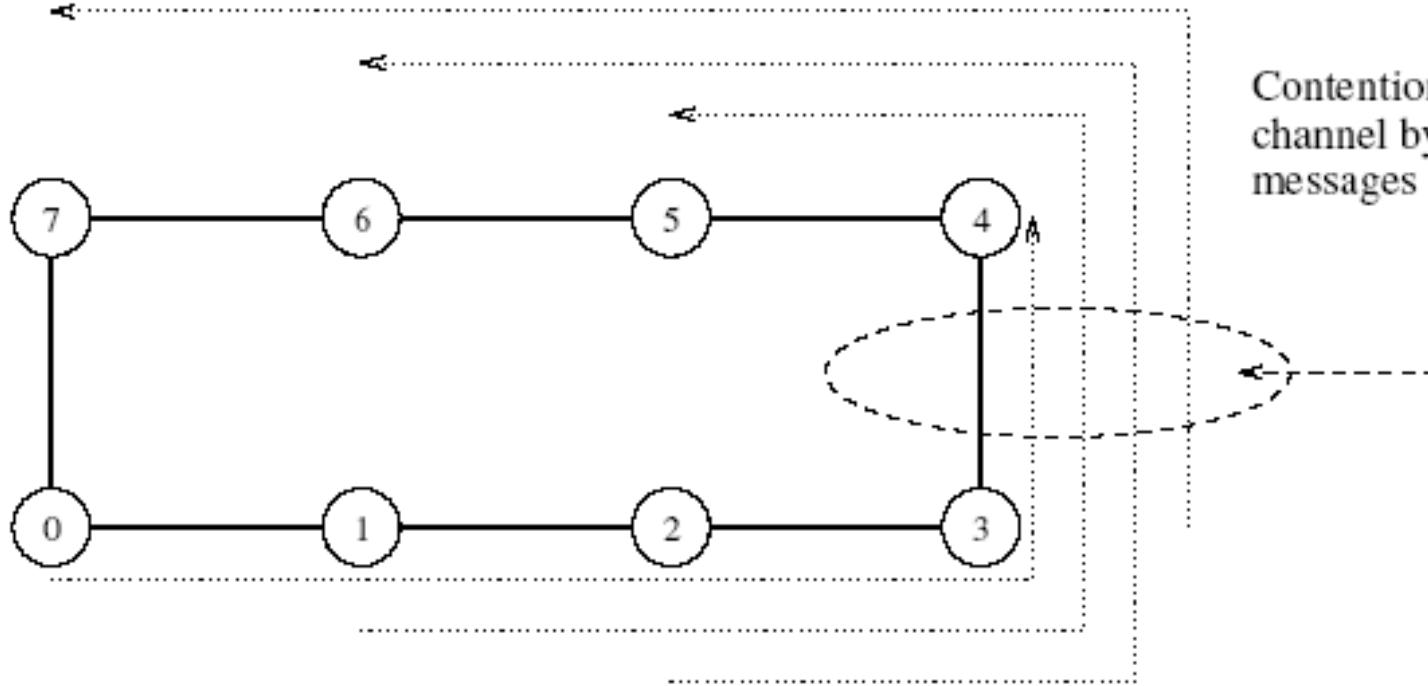
- ◆ Similar communication pattern to all-to-all broadcast, except in the reverse order.
- ◆ On receiving a message, a node must combine it with the local copy of the message that has the same destination as the received message before forwarding the combined message to the next neighbor.

Cost Analysis

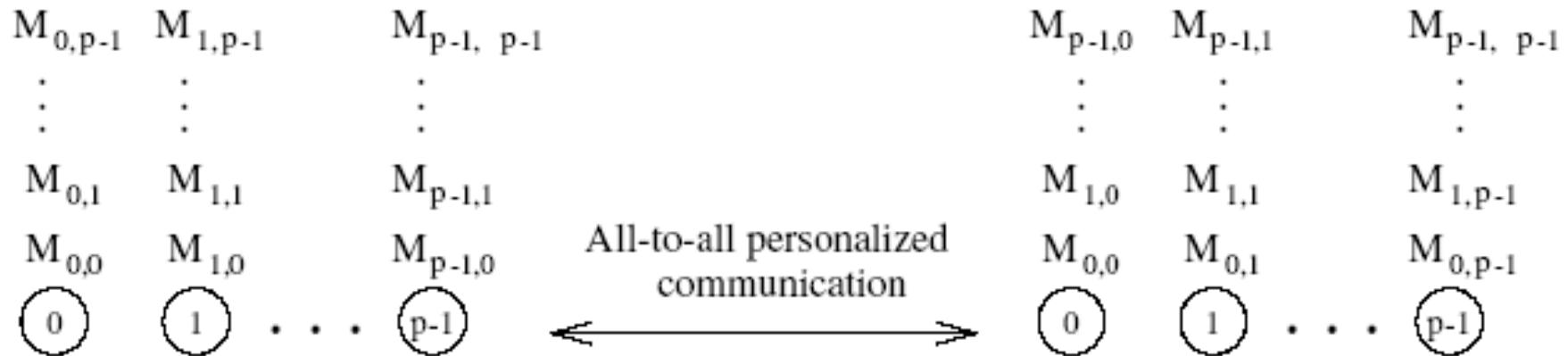
- ◆ On a ring, the time is given by: $(t_s + t_w m)(p-1)$.
- ◆ On a mesh, the time is given by: $2t_s(\sqrt{p} - 1) + t_w m(p-1)$.
- ◆ On a hypercube, we have:

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

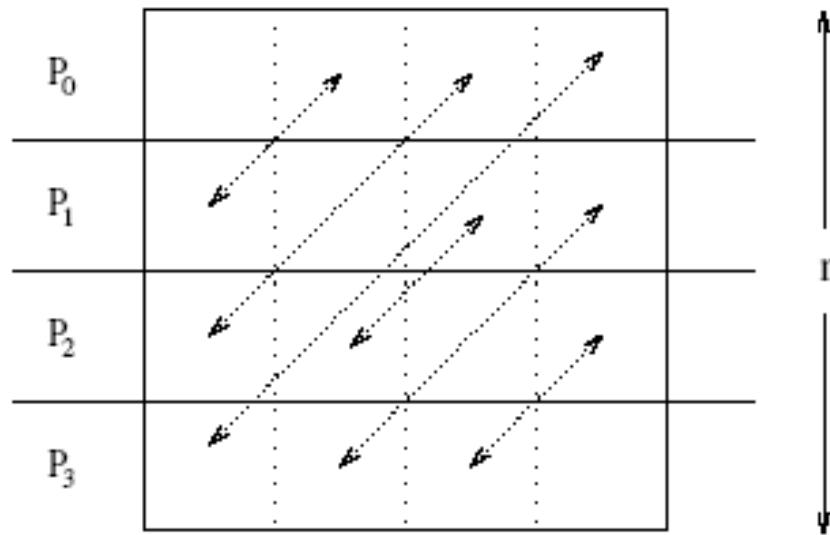
Contention for a channel when the hypercube is mapped onto a ring



All-to-All Personalized Communication



Example

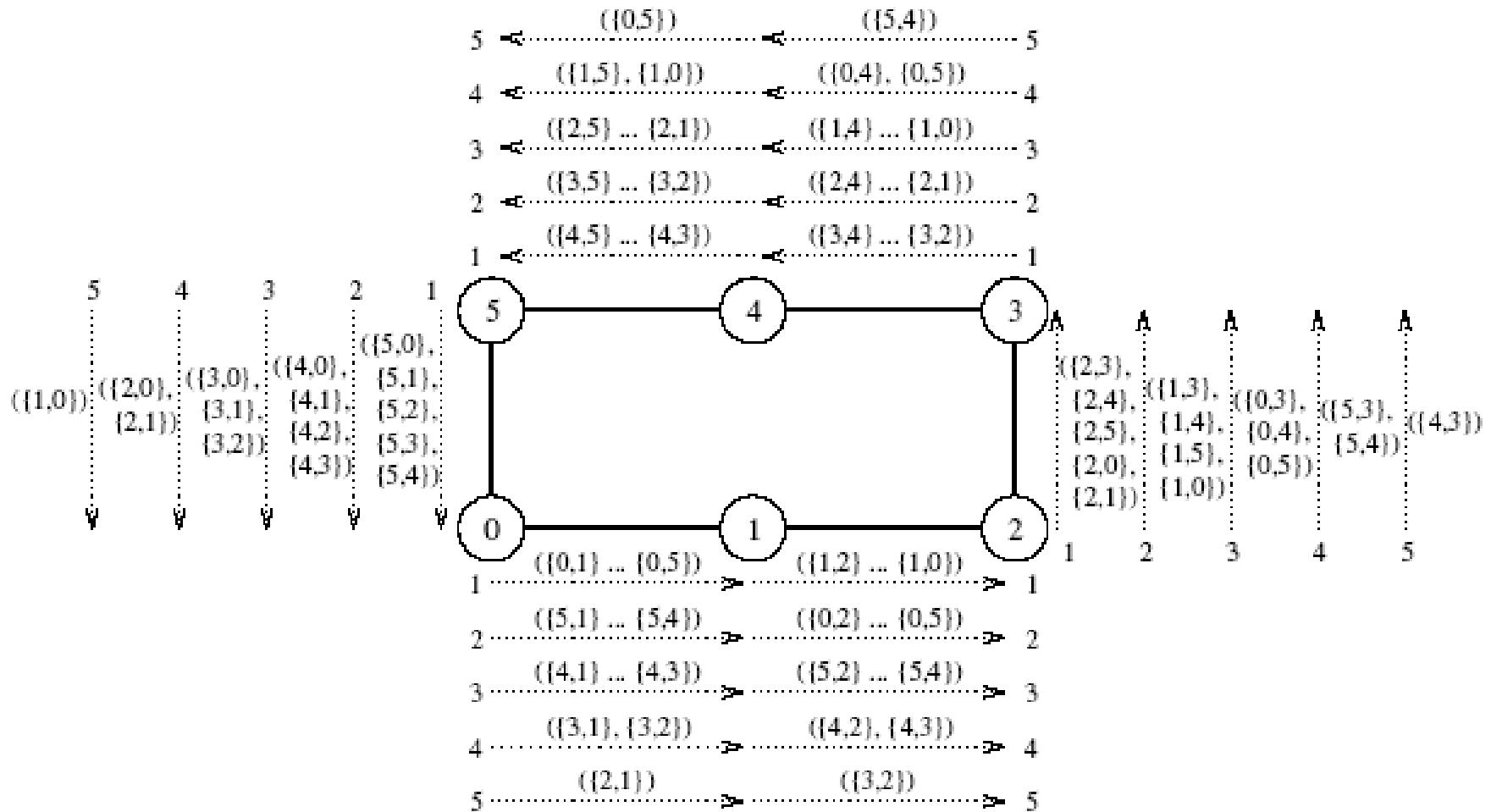


**All-to-all personalized communication
in transposing a 4×4 matrix using four processes.**

Approaches

- ◆ 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



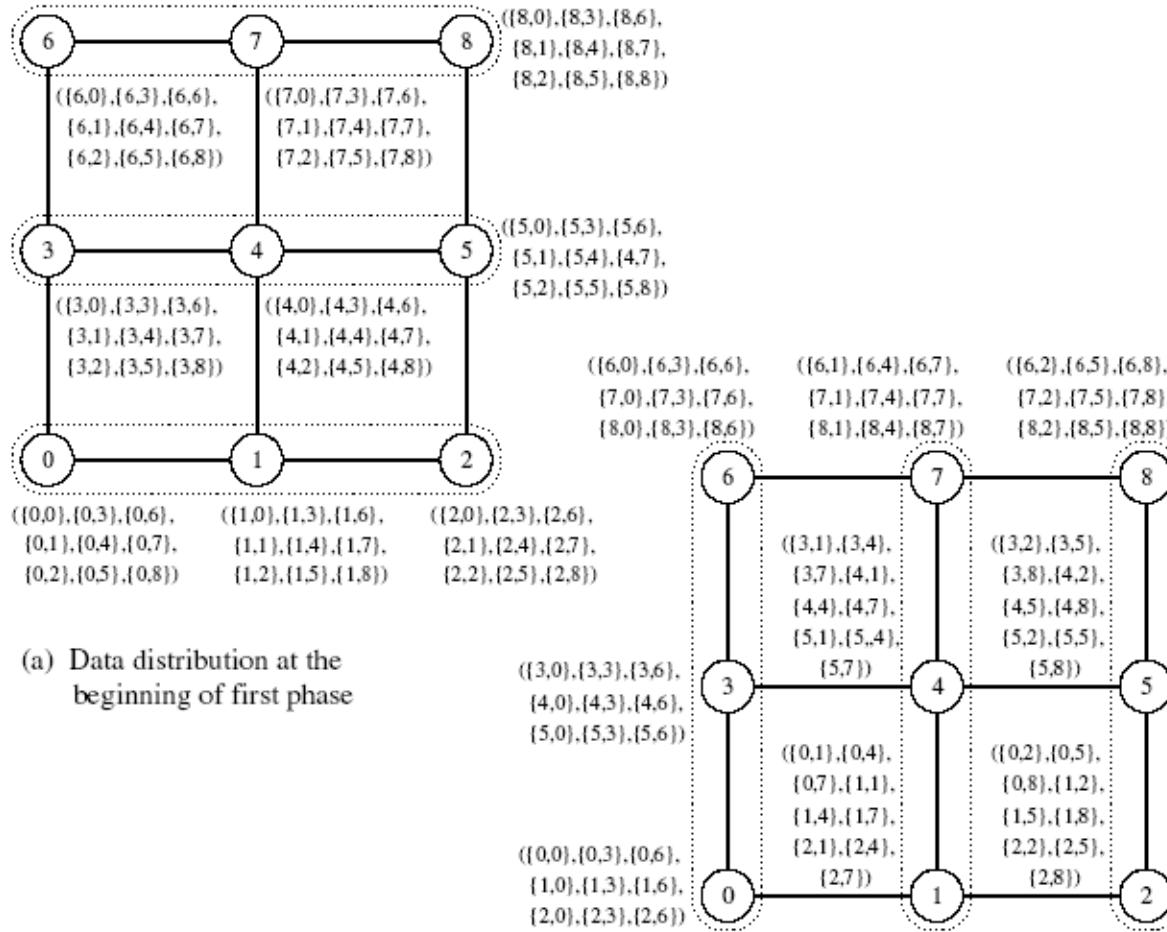
All-to-All Personalized Communication on a Ring: Cost

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

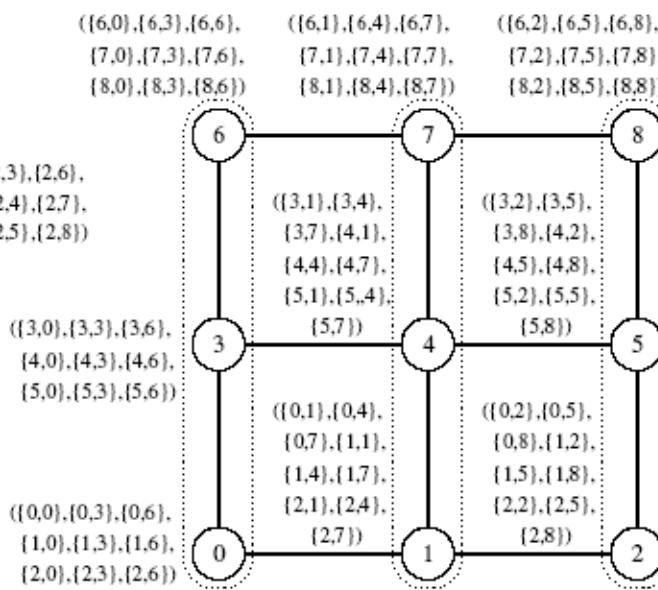
$$\begin{aligned} T &= \sum_{i=1}^{p-1} (t_s + t_w m(p - i)) \\ &= t_s(p - 1) + \sum_{i=1}^{p-1} i t_w m \\ &= (t_s + t_w mp/2)(p - 1). \end{aligned}$$

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

All-to-All Personalized Communication on a Mesh



(a) Data distribution at the beginning of first phase



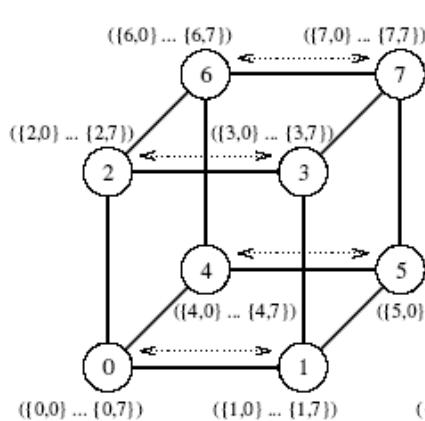
(b) Data distribution at the beginning of second phase

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

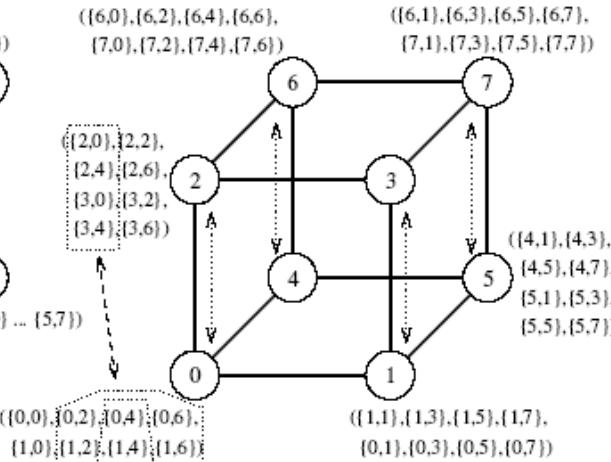
- ◆ Time for the first phase is identical to that in a ring with \sqrt{p} processors, i.e., $(t_s + t_w mp/2)(\sqrt{p} - 1)$.
- ◆ Time in the second phase is identical to the first phase.
- ◆ Therefore, total time is twice of this time, i.e.,

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

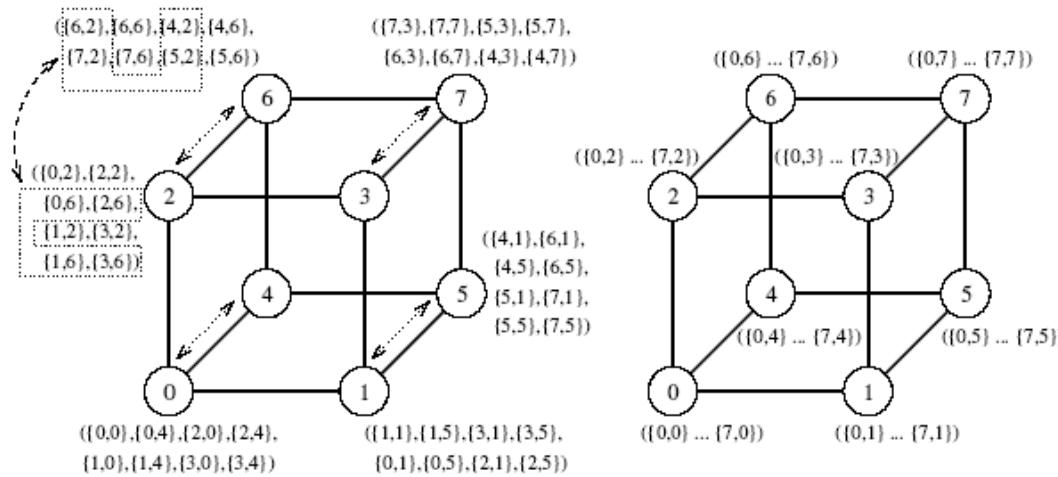
All-to-All Personalized Communication on a Hypercube



(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

◆ Transfer messages $M_{i,j}$ ($0 \leq i, j \leq 2^d - 1$)

- Initially node i ($0 \leq i \leq 2^d - 1$) has messages $M_{i,j}$ ($0 \leq j \leq 2^d - 1$)
- Eventually $M_{i,j}$ will be sent to node j for $j \neq i$

◆ Algorithm

```
procedure ALL_TO_ALL_PERSONAL(d, my_id)
begin
    for i := 1 to d do
        begin
            mask :=  $2^i$ ;
            partner := my_id XOR mask;
            send  $M_{*,k}$  to partner, for  $(k \text{ AND } \text{mask}) \neq (\text{my\_id} \text{ AND } \text{mask})$ ;
            receive  $M_{*,k}$  from partner, for  $(k \text{ AND } \text{mask}) = (\text{my\_id} \text{ AND } \text{mask})$ ;
        end for;
end ALL_TO_ALL_PERSONAL
```

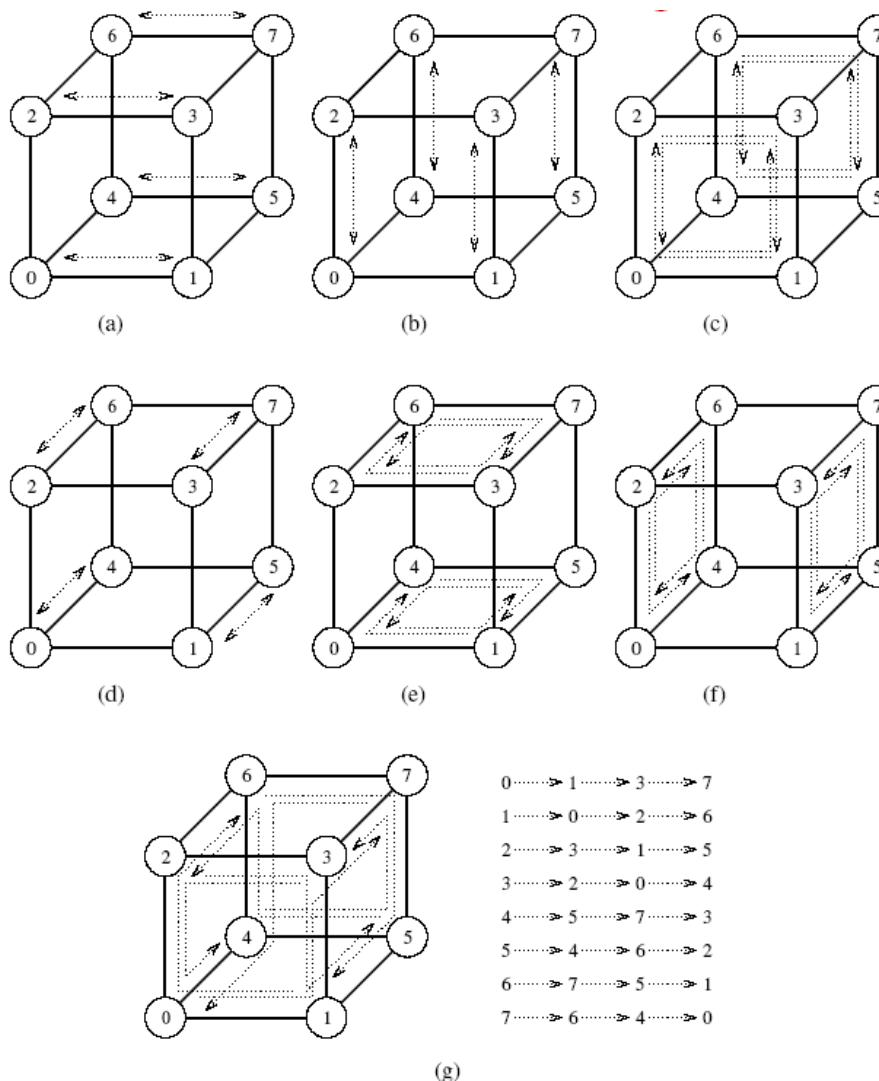
All-to-All Personalized Communication on a Hypercube: Cost

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

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

- ◆ This is not optimal (for long messages)!

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

◆ Algorithm (E-Cube Routing)

```
procedure ALL_TO_ALL_PERSONAL(d, my_id)
begin
    for i := 1 to  $2^d - 1$  do
        begin
            partner := my_id XOR i;
            send  $M_{my\_id, partner}$  to parner;
            receive  $M_{partner, my\_id}$  from partner;
        end for;
    end ALL_TO_ALL_PERSONAL
```

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.

MPI Names of the Various Operations

Operation	MPI Name
One-to-all broadcast	<code>MPI_Bcast</code>
All-to-one reduction	<code>MPI_Reduce</code>
All-to-all broadcast	<code>MPI_Allgather</code>
All-to-all reduction	<code>MPI_Reduce_scatter</code>
All-reduce	<code>MPI_Allreduce</code>
Gather	<code>MPI_Gather</code>
Scatter	<code>MPI_Scatter</code>
All-to-all personalized	<code>MPI_Alltoall</code>