Introduction to Parallel & Distributed Computing

Programming with MPI Communications

Lecture 7, Spring 2024

Instructor: 罗国杰

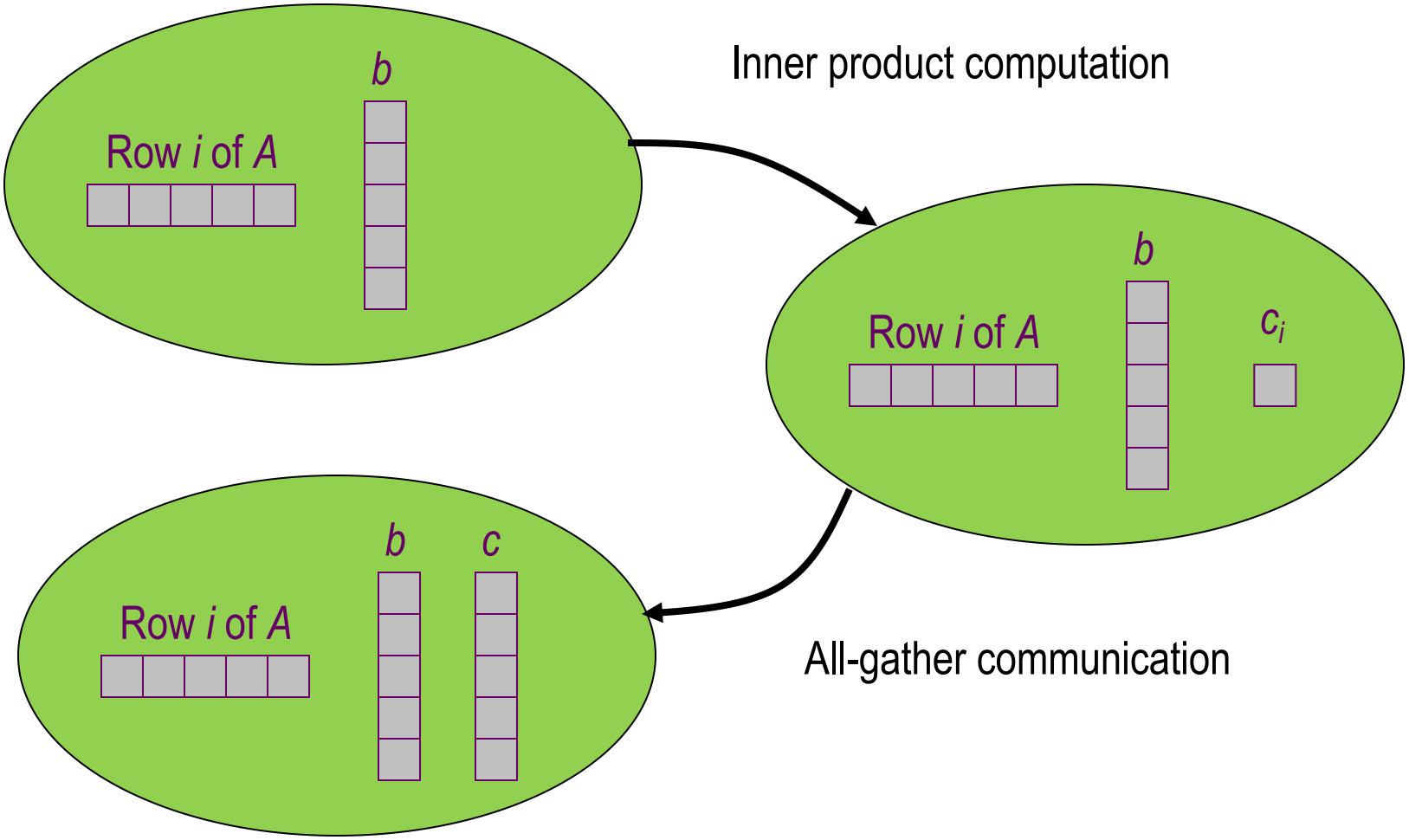
gluo@pku.edu.cn

Example: Matrix-Vector Multiplication

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

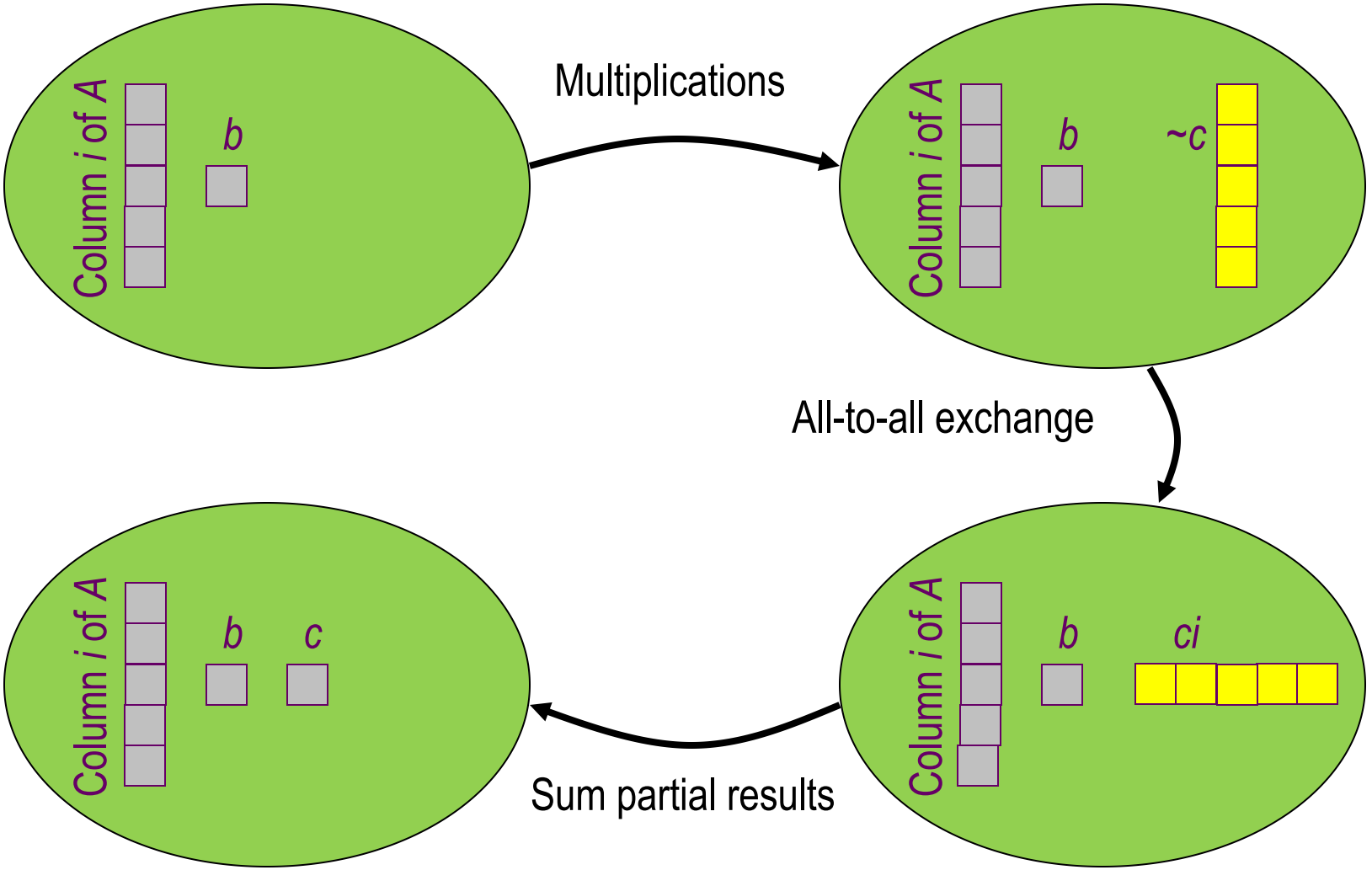
Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.

Phases of Parallel Algorithm *(Rowwise)*



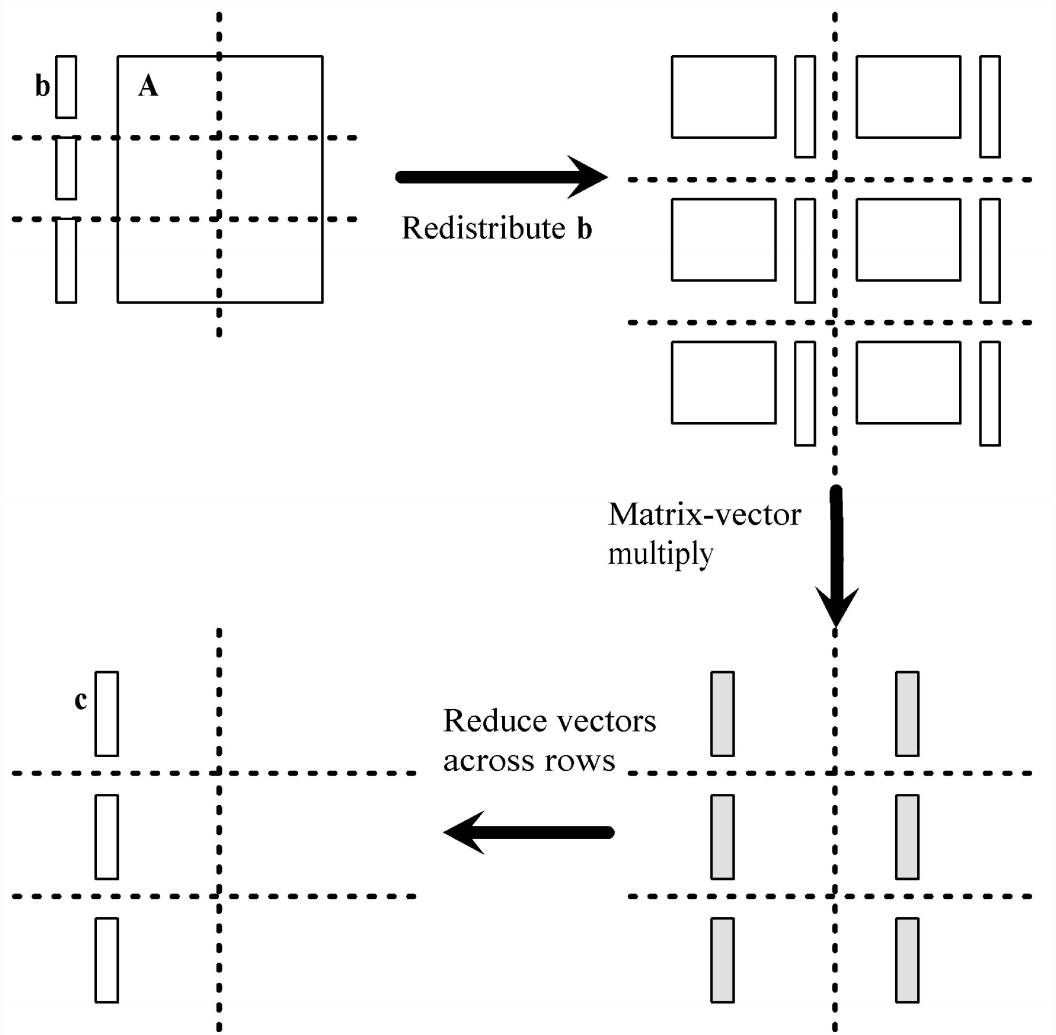
Michael J.Quinn, "Parallel Programming in C with MPI and OpenMP," 2003.

Phases of Parallel Algorithm (Columnwise)



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

Phases of Parallel Algorithm (Checkerboard)



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

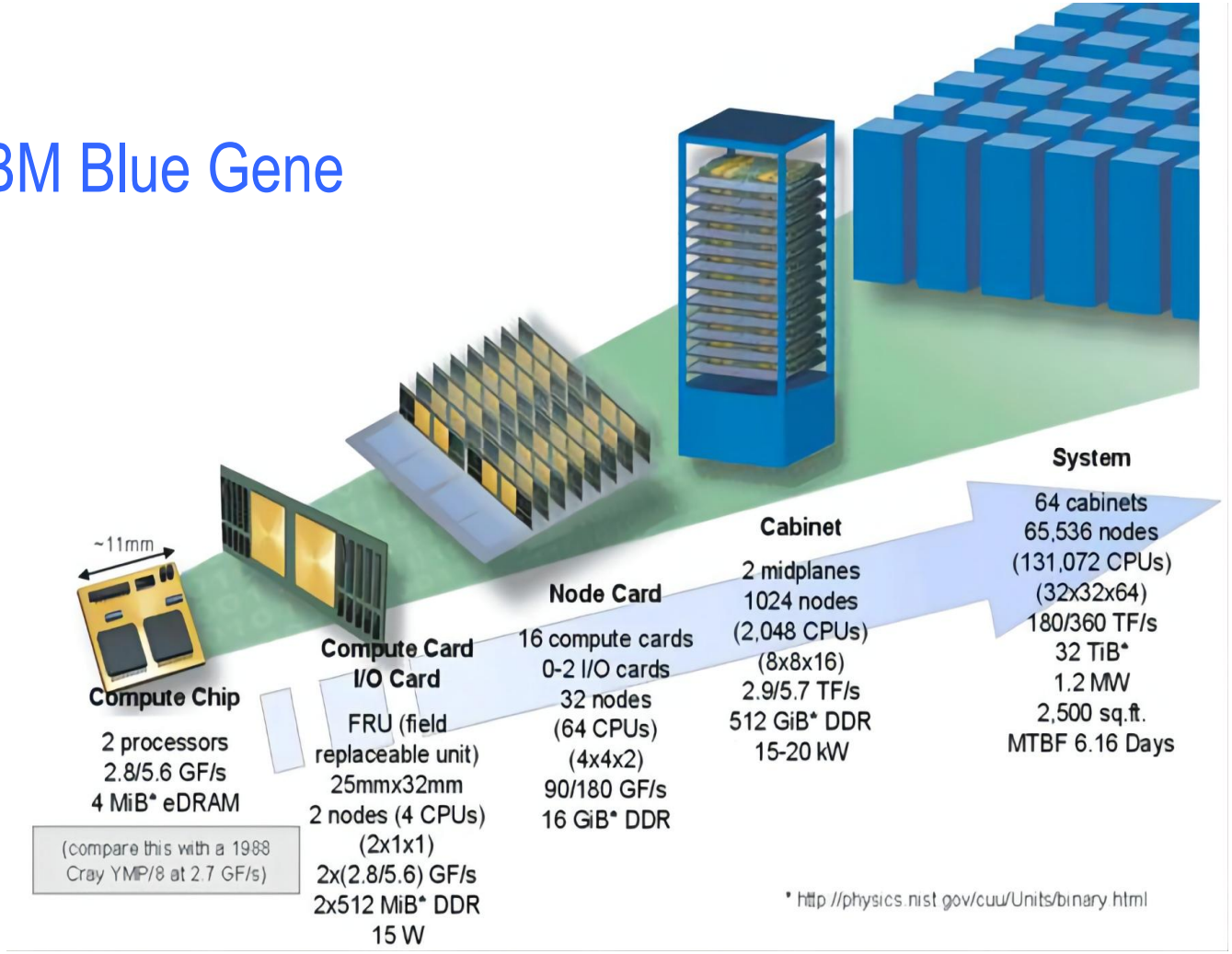
Collective Communications

	Ring	2-D Mesh	Hypercube
One-to-All Broadcast			
All-to-One Reduction			
All-to-All Broadcast			
All-to-All Reduction			
Scatter			
Gather			
All-to-All Personalized Communication			

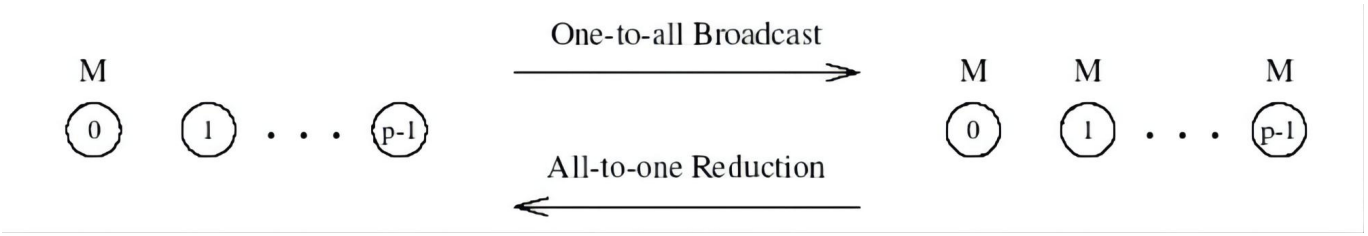
Dual of a communication operation

Multi-Dimensional Structure

IBM Blue Gene

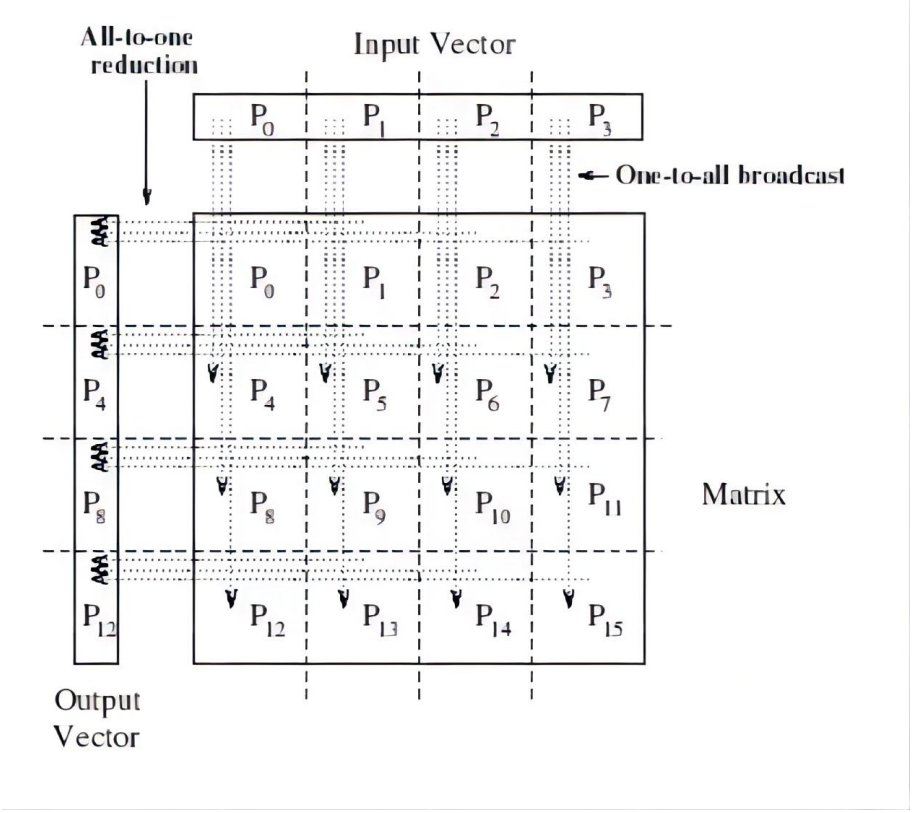


One-to-All Broadcast and All-to-One Reduction



A. Grama et al., "Introduction to Parallel Computing," Addison Wesley, 2003

Example



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

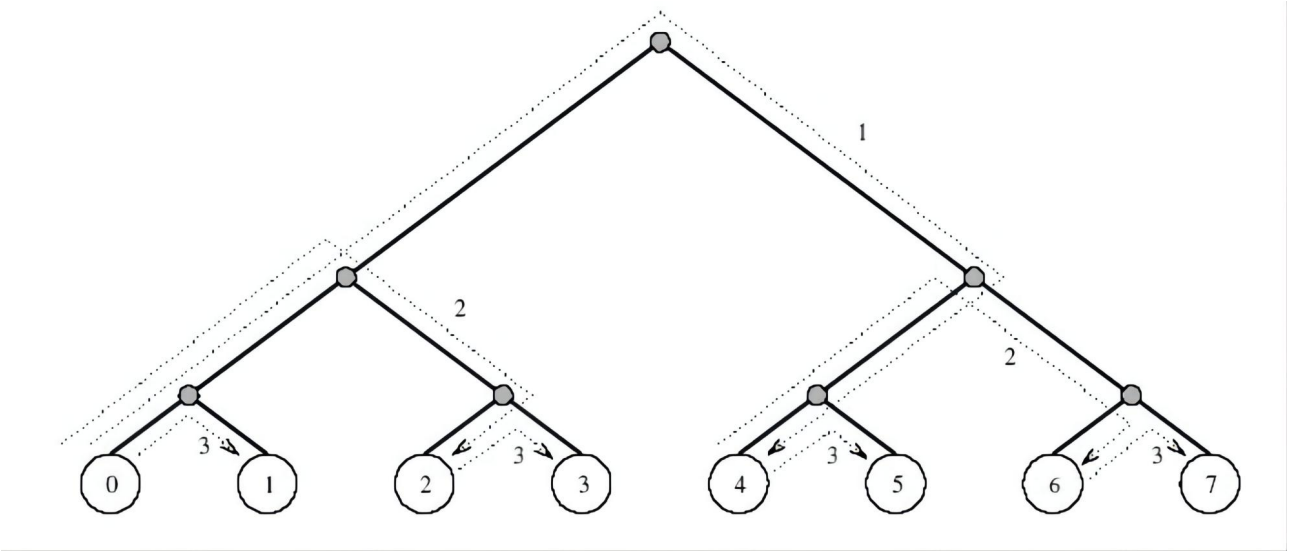
A. Grama et al., "Introduction to Parallel Computing," Addison Wesley, 2003

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.**

A. Grama et al., "Introduction to Parallel Computing," Addison Wesley, 2003

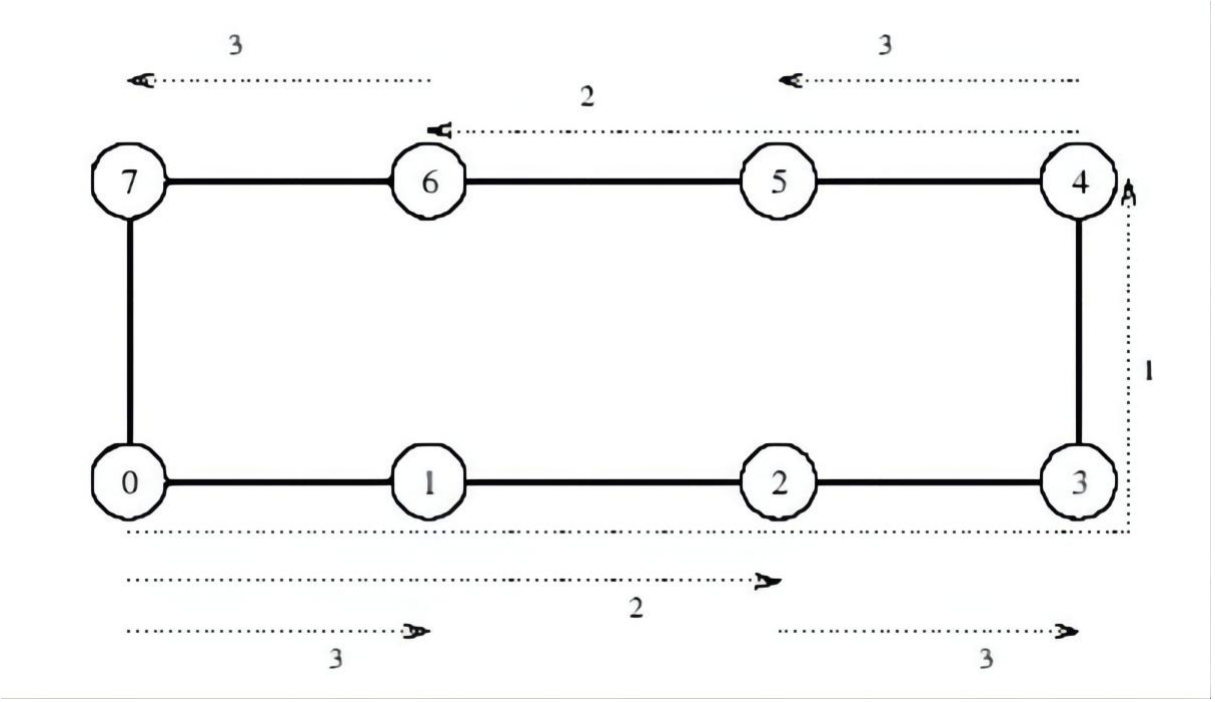
Broadcast and Reduction on a Balanced Binary Tree



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

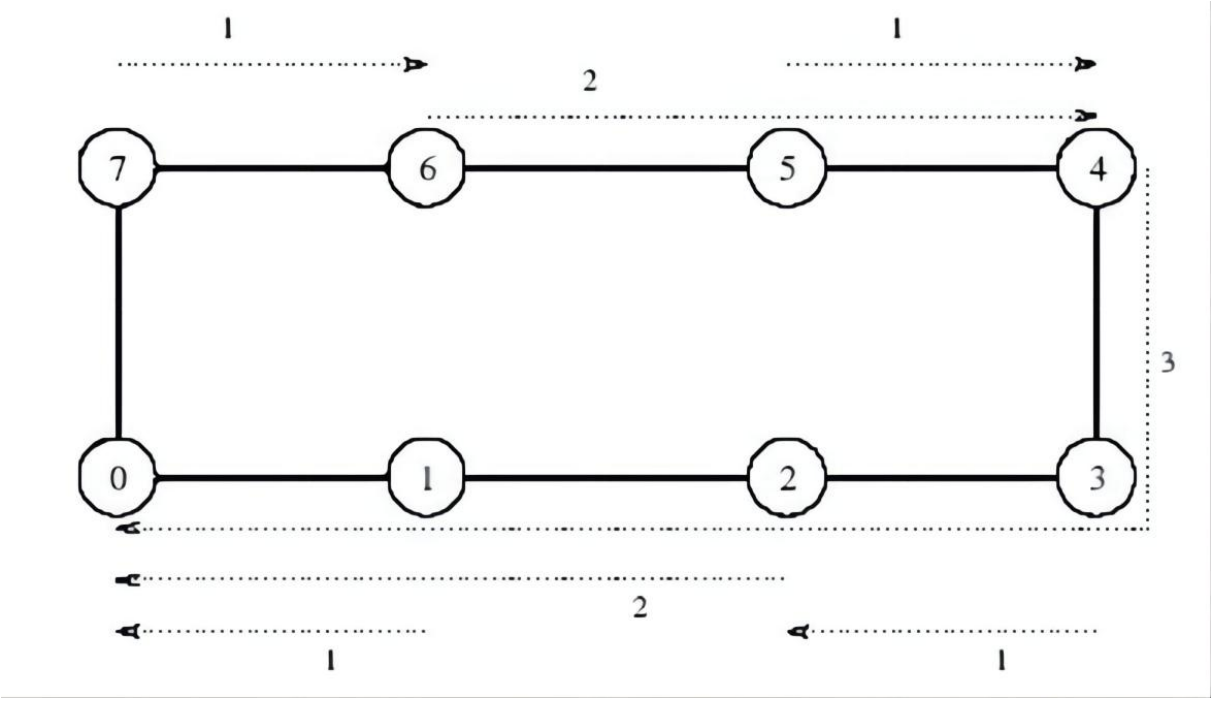
A. Grama et al., "Introduction to Parallel Computing," Addison Wesley, 2003

One-to-All Broadcast on Ring



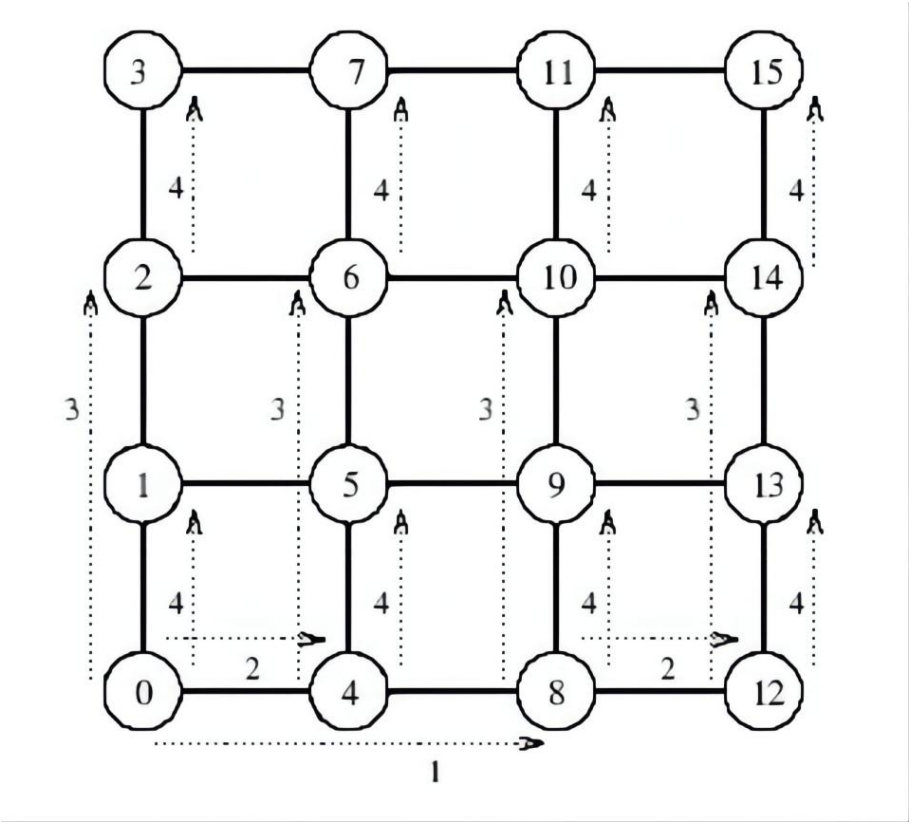
A. Grama et al., "Introduction to Parallel Computing," Addison Wesley, 2003

All-to-One Reduction



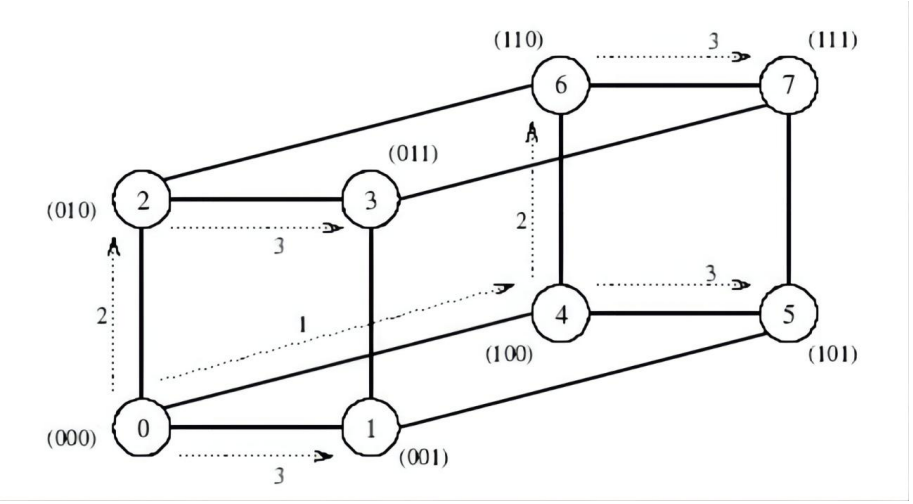
A. Grama et al., "Introduction to Parallel Computing," Addison Wesley, 2003

Broadcast and Reduction on a Mesh: Example



A. Grama et al., "Introduction to Parallel Computing," Addison Wesley, 2003

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.

A. Grama et al., "Introduction to Parallel Computing," Addison Wesley, 2003

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.

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 reply on randomized routing to alleviate hot-spots and contention on the network. Its 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

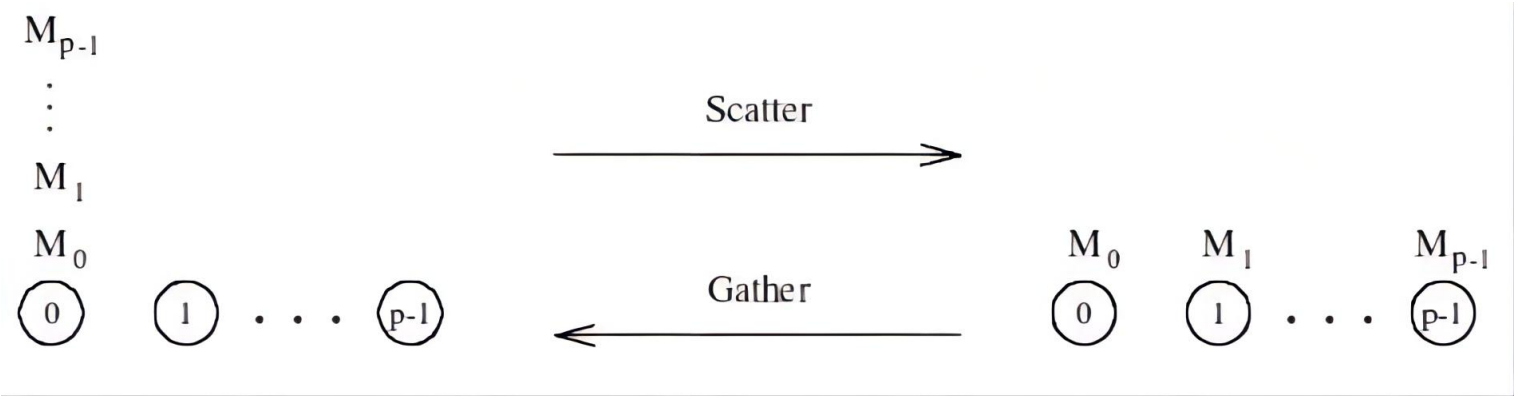
Cost Analysis of One-to-All Broadcast and All-to-One Reduction

- **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:**

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

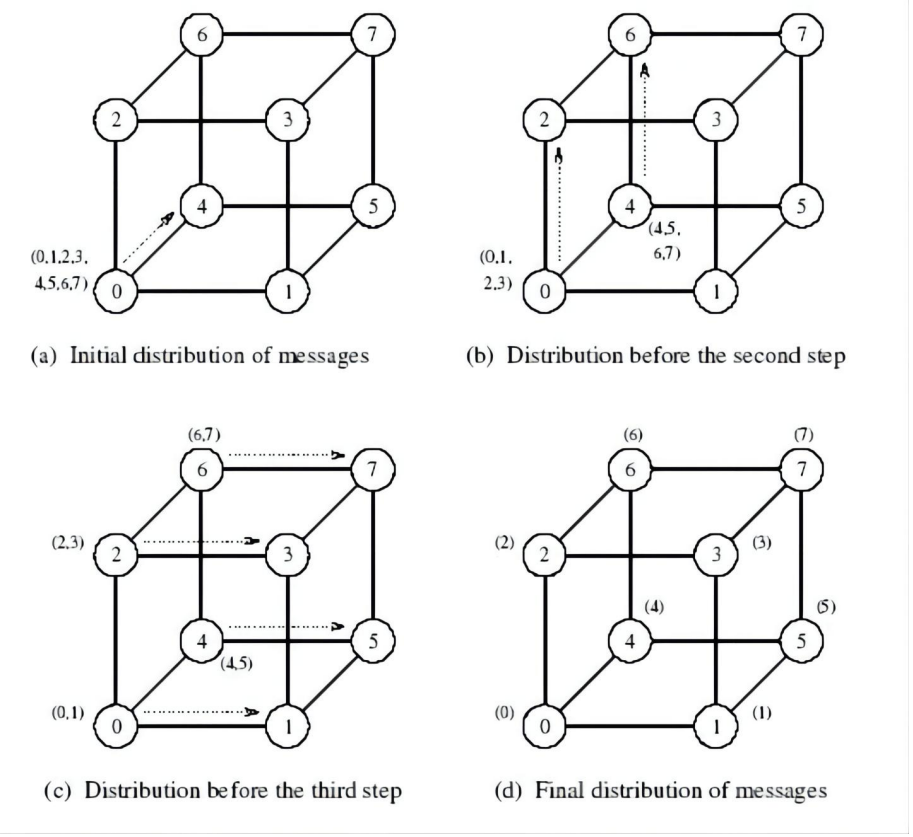
A. Grama et al., "Introduction to Parallel Computing," Addison Wesley, 2003

Gather and Scatter Operations



A. Grama et al., "Introduction to Parallel Computing," Addison Wesley, 2003

Example of the Scatter Operation



The scatter operation on an eight-node hypercube.

A. Grama et al., "Introduction to Parallel Computing," Addison Wesley, 2003

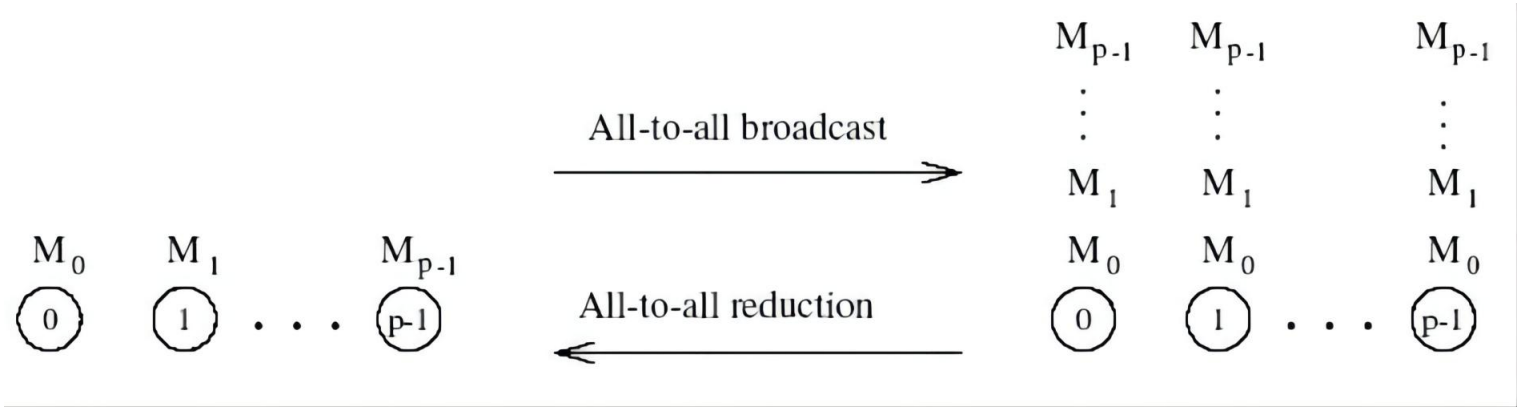
Cost of Scatter and Gather

- **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:**

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

A. Grama et al., "Introduction to Parallel Computing," Addison Wesley, 2003

All-to-All Broadcast and Reduction



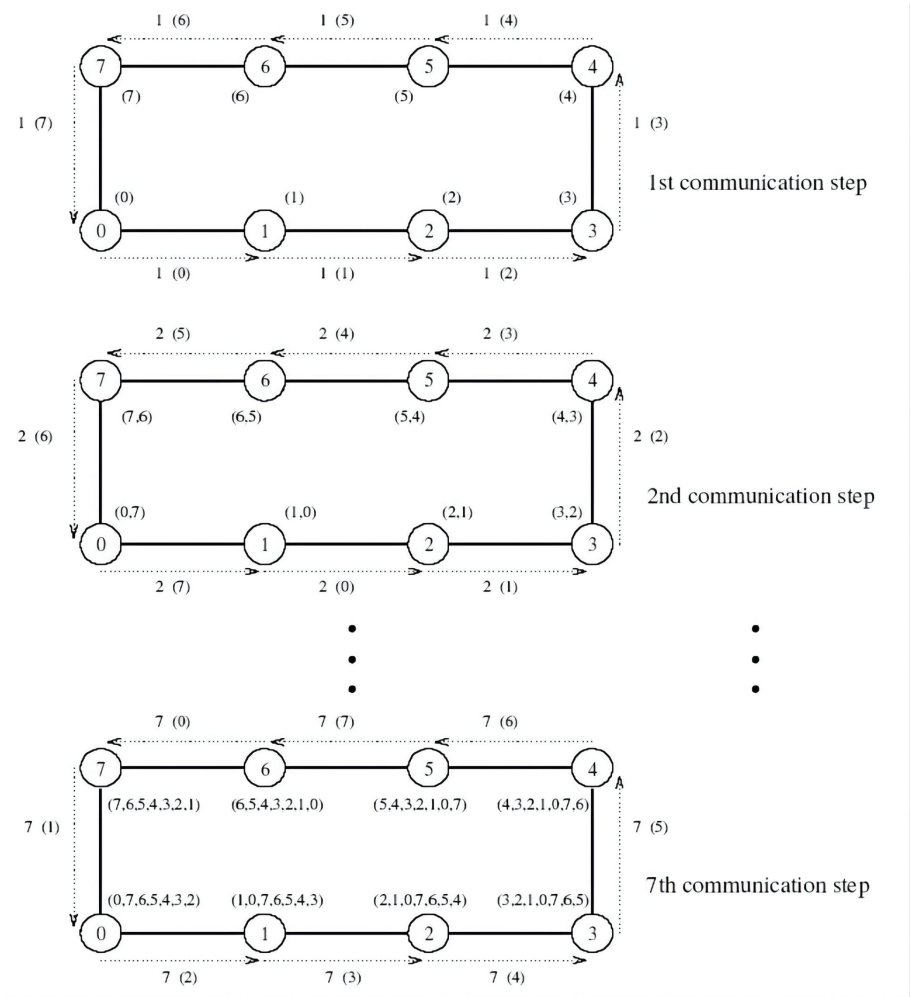
A. Grama et al., "Introduction to Parallel Computing," Addison Wesley, 2003

Approaches

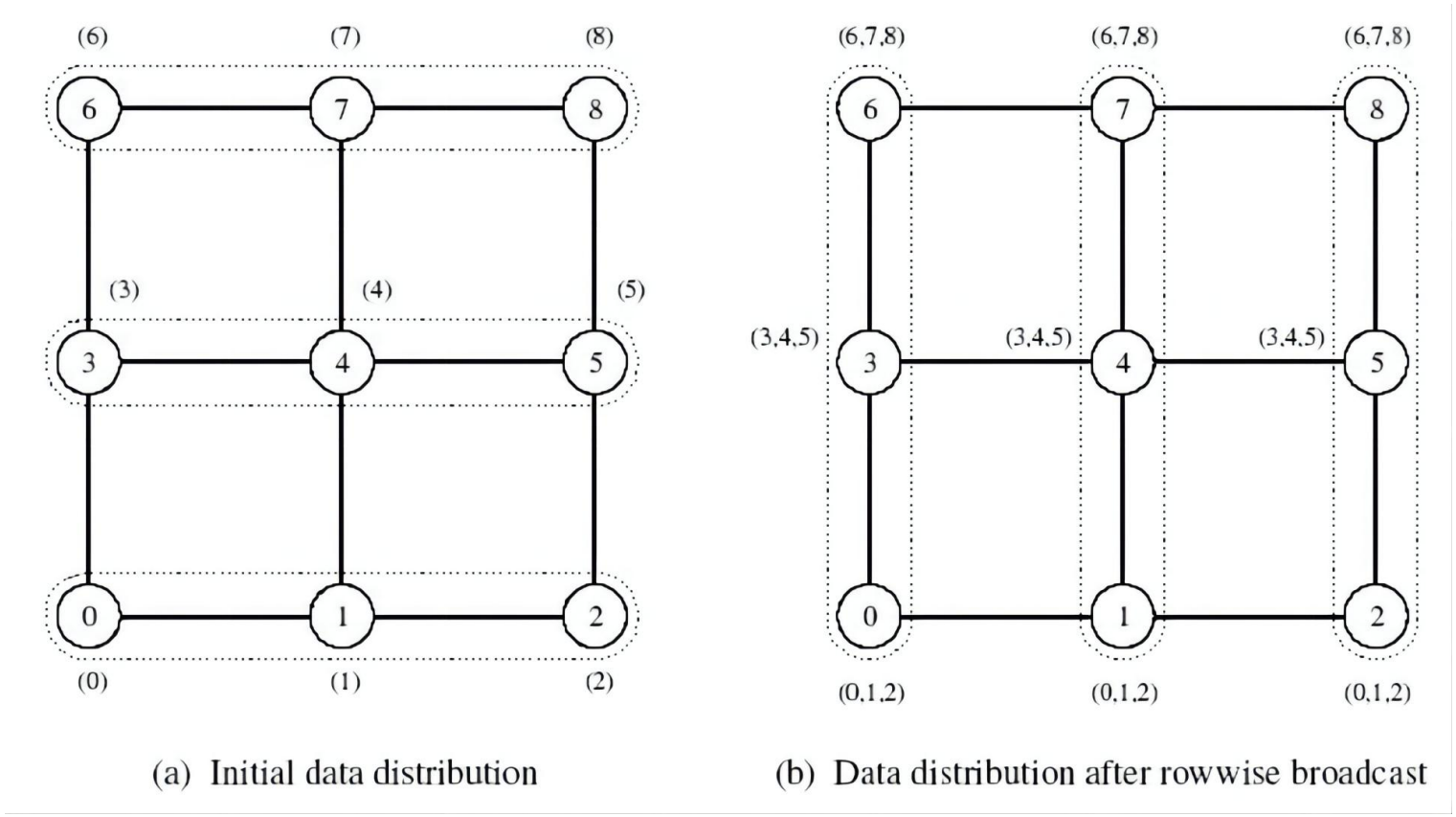
- **Simplest approach**
  - **Perform *p* one-to-all broadcasts.**
  - **This is not the most efficient way, though.**
- **Better approach**
  - **Each node first sends to one of its neighbors the data it needs to broadcast.**
  - **In subsequent steps, it forwards the data received from one of its neighbors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

A. Grama et al., "Introduction to Parallel Computing," Addison Wesley, 2003

All-to-All Broadcast and Reduction on a Ring

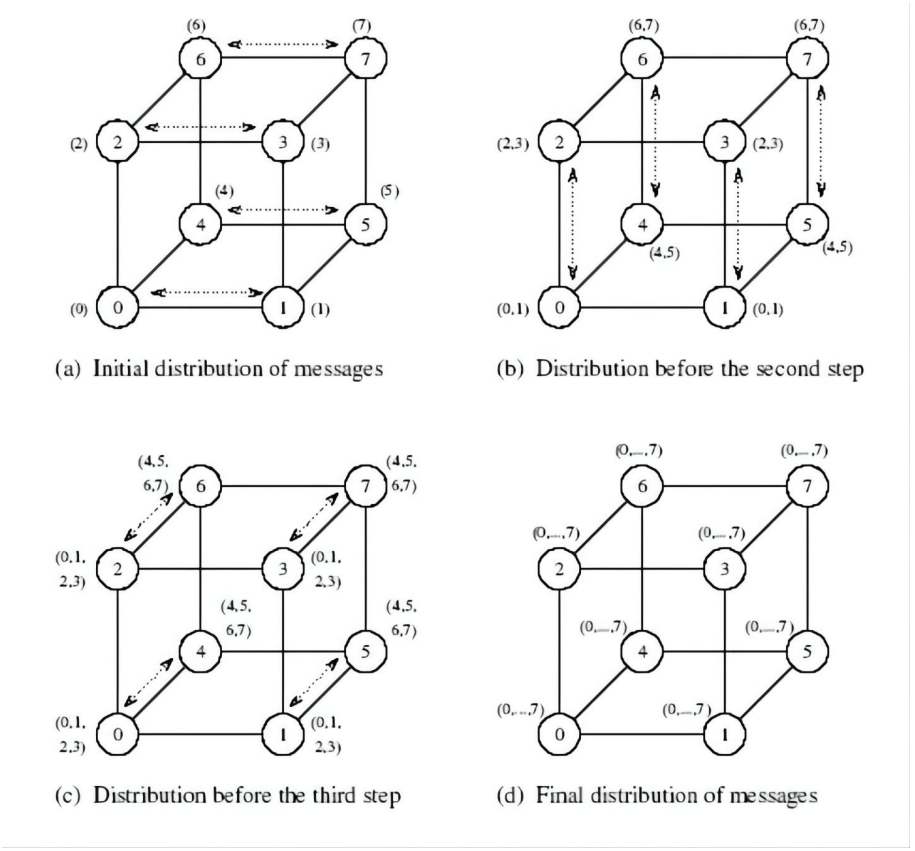


All-to-all Broadcast on a Mesh



A. Grama et al., "Introduction to Parallel Computing," Addison Wesley, 2003

All-to-all broadcast on a Hypercube



A. Grama et al., "Introduction to Parallel Computing," Addison Wesley, 2003

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.

A. Grama et al., "Introduction to Parallel Computing," Addison Wesley, 2003

Cost Analysis

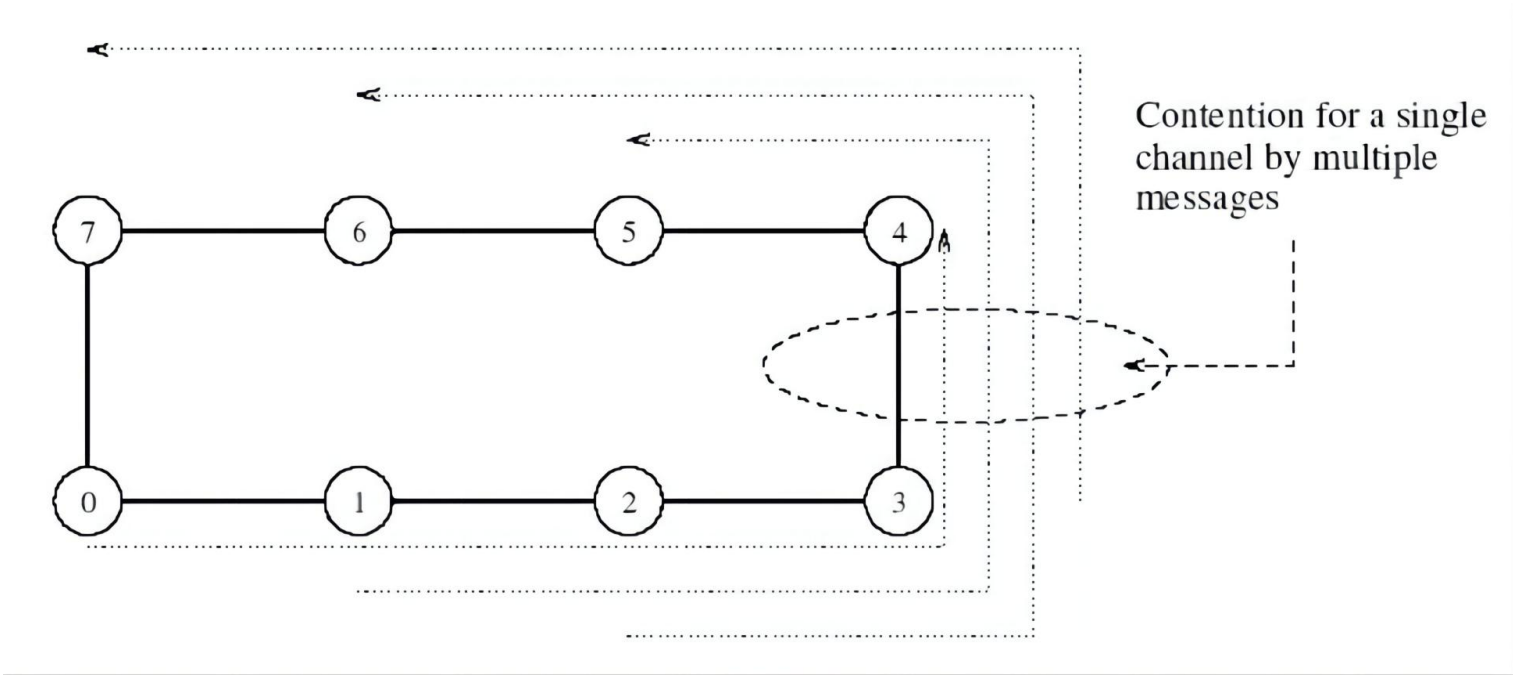
- **On a ring, the time is given by:** $(t_s + t_w m)(p-1)$.
- **On a mesh, the time is given by:** $2t_s(\sqrt{p} - 1) + t_w m(p-1)$.
- **On a hypercube, we have:**

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

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

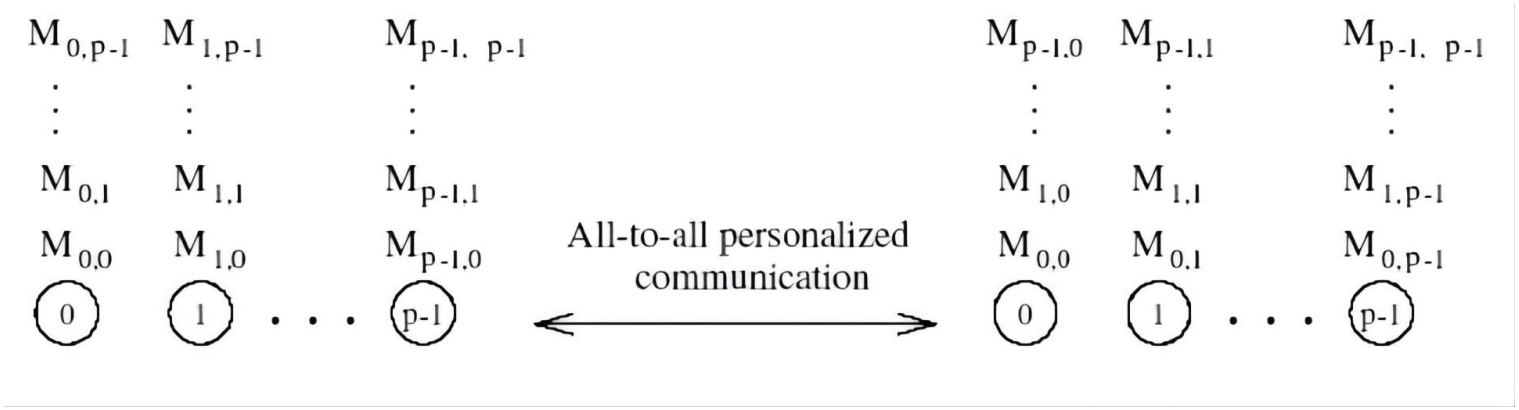
A. Grama et al., "Introduction to Parallel Computing," Addison Wesley, 2003

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



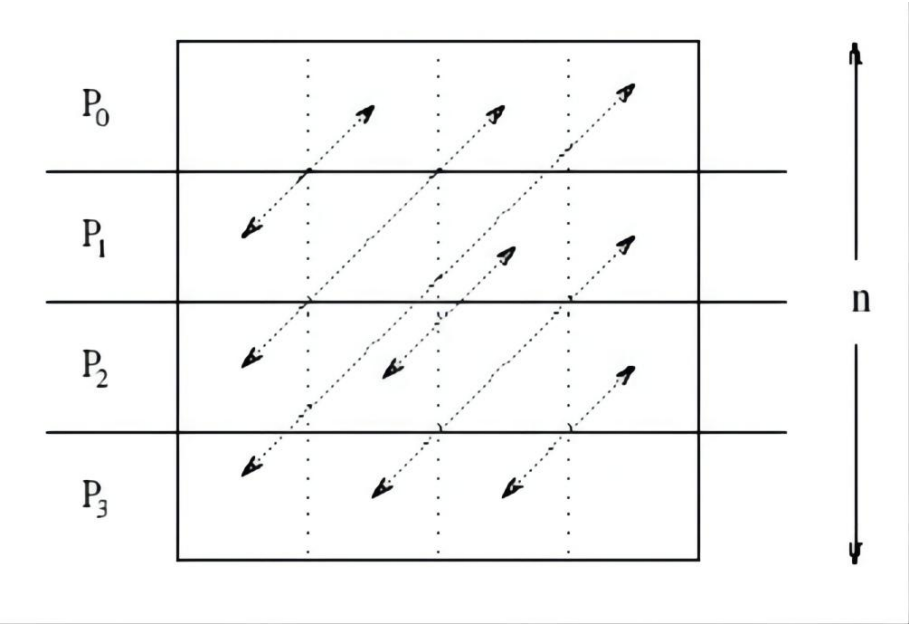
A. Grama et al., "Introduction to Parallel Computing," Addison Wesley, 2003

All-to-All Personalized Communication



A. Grama et al., "Introduction to Parallel Computing," Addison Wesley, 2003

Example



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

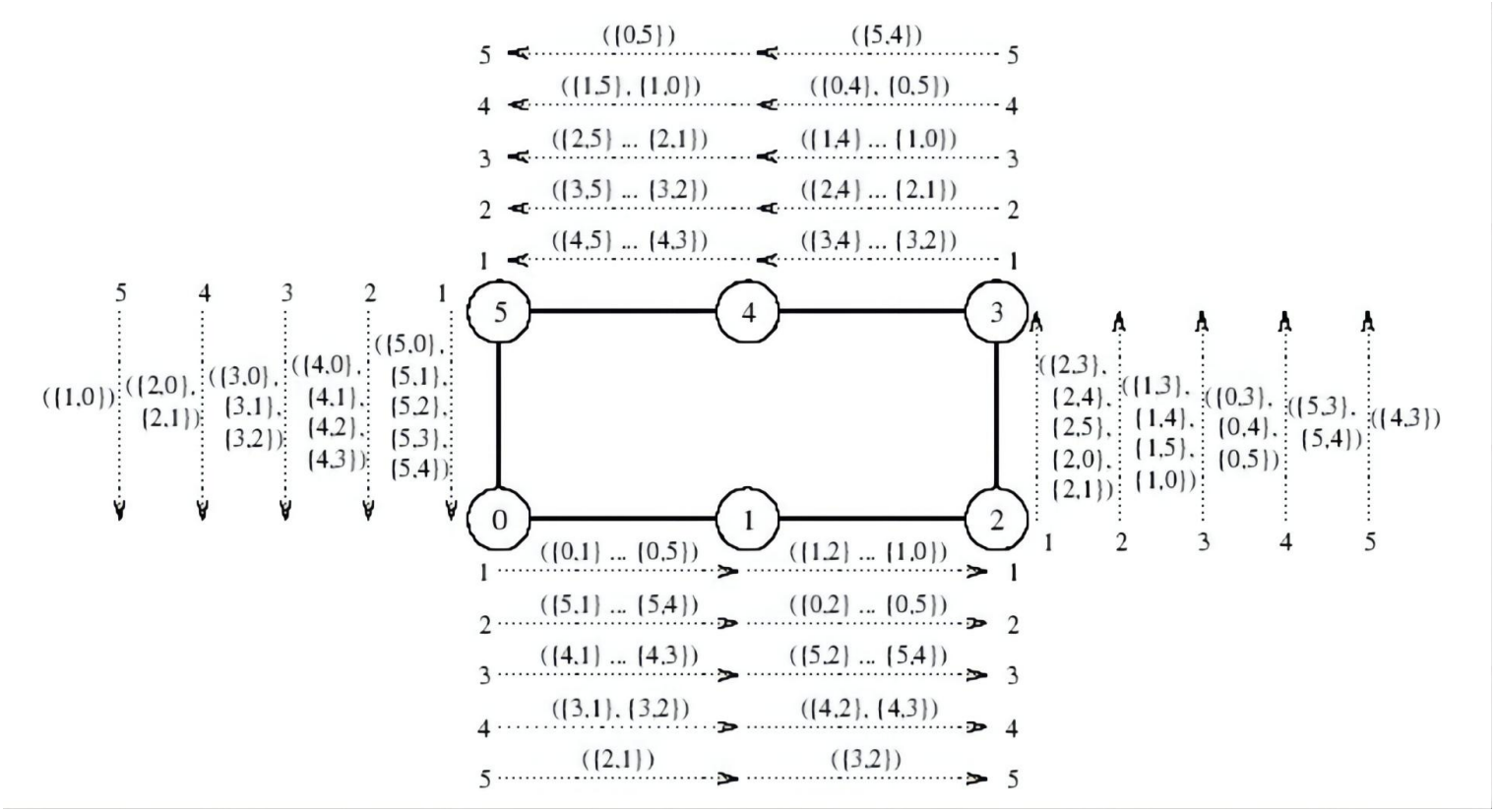
A. Grama et al., "Introduction to Parallel Computing," Addison Wesley, 2003

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.**

A. Grama et al., "Introduction to Parallel Computing," Addison Wesley, 2003

All-to-All Personalized Communication on a Ring



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

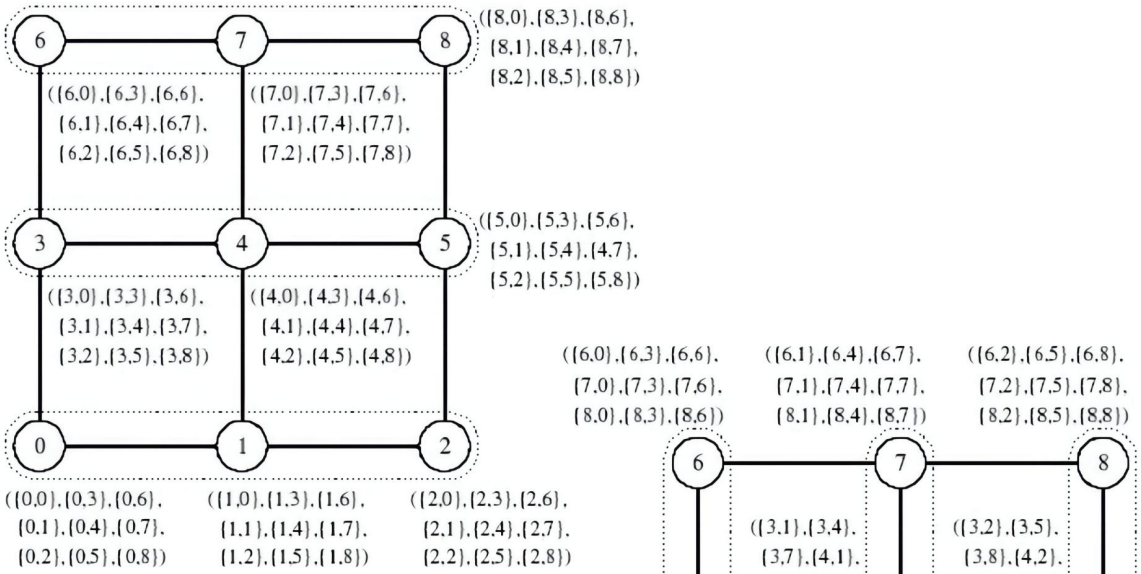
- **We have $p - 1$ steps in all.**
- **In step i, the message size is $m(p - i)$.**
- **The total time is given by:**

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

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

A. Grama et al., "Introduction to Parallel Computing," Addison Wesley, 2003

All-to-All Personalized Communication on a Mesh



A. Grama et al., "Introduction to Parallel Computing," Addison Wesley, 2003

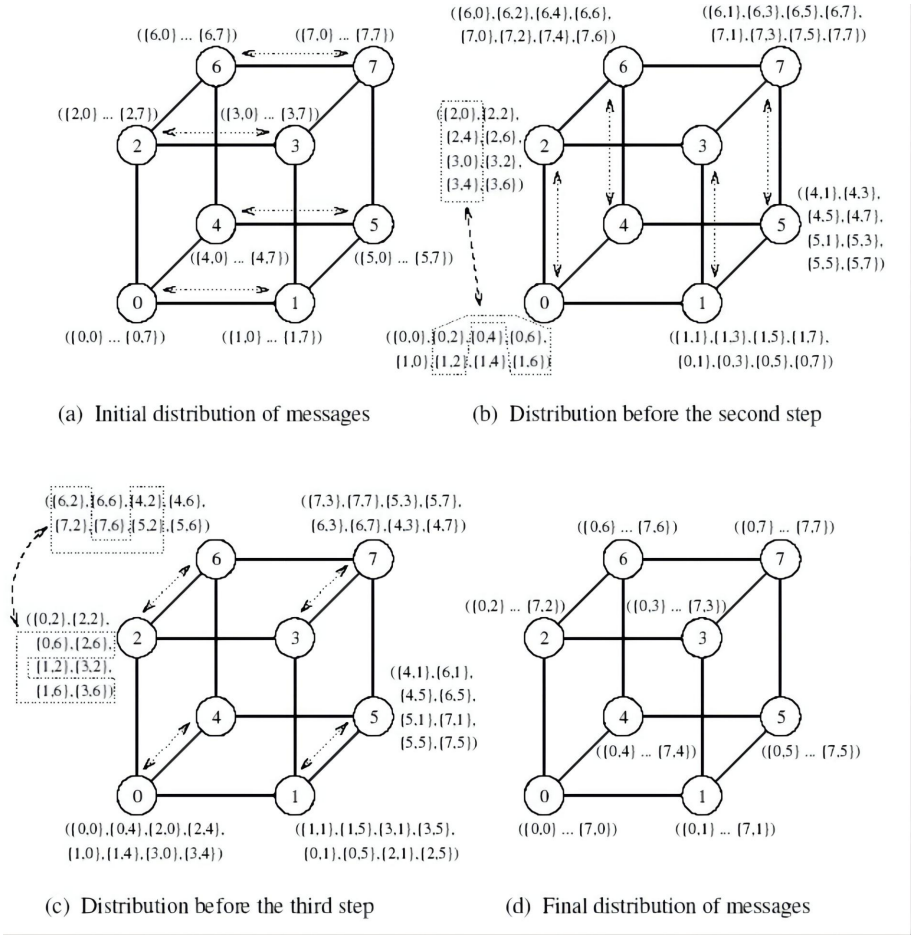
All-to-All Personalized Communication on a Mesh: Cost

- **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.,**

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

A. Grama et al., "Introduction to Parallel Computing," Addison Wesley, 2003

All-to-All Personalized Communication on a Hypercube



All-to-All Personalized Communication on a Hypercube

- **Transfer messages $M_{i,j}$ $(0 \le i, j \le 2^d - 1)$**
  - **Initially node i $(0 \le i \le 2^d - 1)$ has messages $M_{i,j}$ $(0 \le j \le 2^d - 1)$**
  - **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 AND mask) ≠ (my_id AND mask);
    receive M*,k from partner,   for (k AND mask) = (my_id AND mask);
  end for;
end ALL_TO_ALL_PERSONAL
```

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

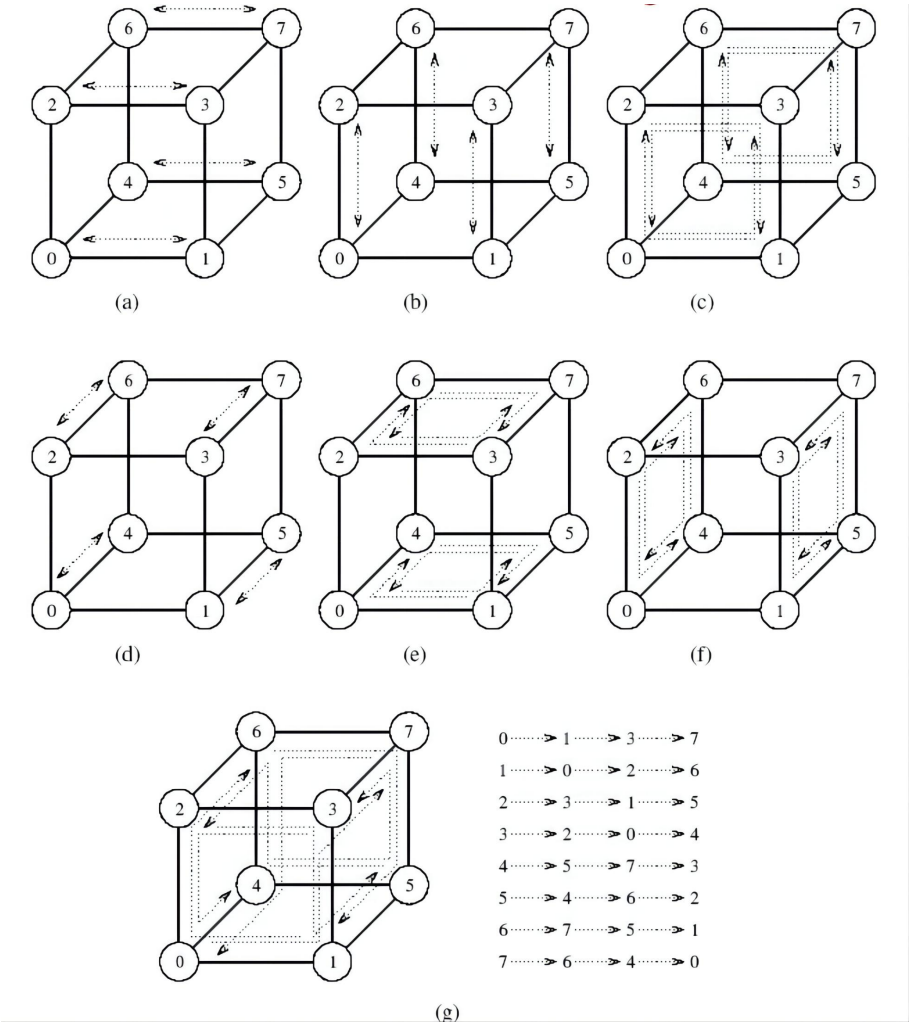
- **We have *log p* iterations and *mp/2* words are communicated in each iteration. Therefore, the cost is:**

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

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

A. Grama et al., "Introduction to Parallel Computing," Addison Wesley, 2003

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
```

A. Grama et al., "Introduction to Parallel Computing," Addison Wesley, 2003

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.

A. Grama et al., "Introduction to Parallel Computing," Addison Wesley, 2003

Collective Communications

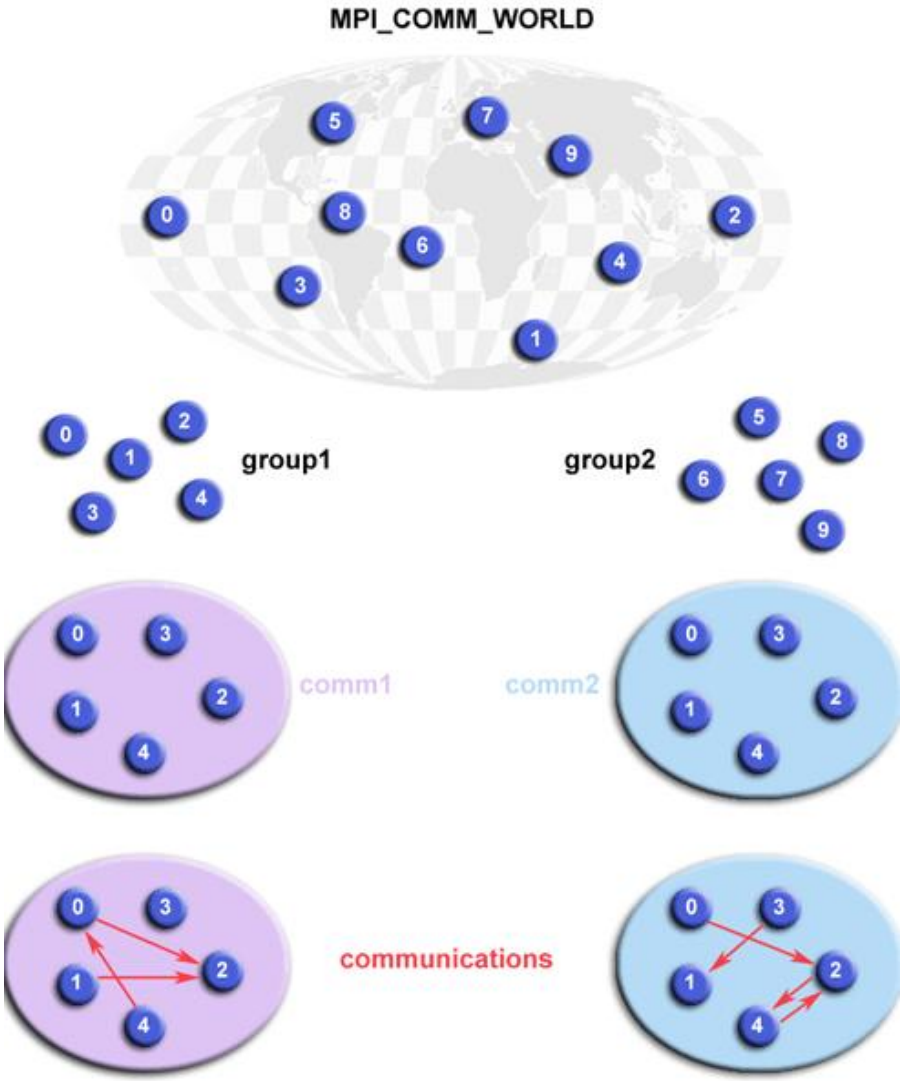
	Ring	2-D Mesh	Hypercube
One-to-All Broadcast / *All-to-One Reduction*	$(t_s+t_wm)\log(p)$	$(t_s+t_wm)\log(p)$	$(t_s+t_wm)\log(p)$
All-to-All Broadcast / *All-to-All Reduction*	$(t_s+t_wm)(p-1)$	$2t_s(\sqrt{p}-1)+t_wm(p-1)$	$t_s\log(p)+t_wm(p-1)$
Scatter / *Gather*	$t_s\log(p)+t_wm(p-1)$	$t_s\log(p)+t_wm(p-1)$	$t_s\log(p)+t_wm(p-1)$
All-to-All Personalized Communication	$(t_s+t_wmp/2)(p-1)$	$(2t_s+t_wmp)(\sqrt{p}-1)$	$(t_w+t_wmp/2)\log(p)$ or $(t_s+t_wm)(p-1)$

MPI Names of the Various Operations

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

A. Grama et al., "Introduction to Parallel Computing," Addison Wesley, 2003

Groups and Communicators



- **Related MPI functions**
  - **Form new group as a subset of global group using** *MPI_Group_incl*
  - **Create new communicator for new group using** *MPI_Comm_create*
  - **Determine new rank in new communicator using** *MPI_Comm_rank*
  - **Conduct communications using any MPI message passing routine**
  - **When finished, free up new communicator and group (optional) using** *MPI_Comm_free* **and** *MPI_Group_free*

B. Barney, "Message Passing Interface (MPI)", LLNL.

Communicators

- **A communicator defines a *communication domain* - a set of processes that are allowed to communicate with each other**
- **Information about communication domains is stored in variables of type `MPI_Comm`**
- **Communicators are used as arguments to all message transfer MPI routines**
- **A process can belong to many different (possibly overlapping) communication domains**
- **MPI defines a default communicator called `MPI_COMM_WORLD` which includes all the processes**

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

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

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**

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

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 */
```

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

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, periodic,
                 1, &cart_comm);
```

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

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

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

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 */
```

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

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 */
```

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

MPI_Comm_split

- **Partitions the processes of a communicator into one or more subgroups**
- **Constructs a communicator for each subgroup**
- **Allows processes in each subgroup to perform their own collective communications**
- **Needed for columnwise scatter and rowwise reduce**

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

Header for MPI_Comm_split

```
int MPI_Comm_split (
    MPI_Comm old_comm,
        /* In - Existing communicator */

    int partition, /* In - Partition number */

    int new_rank,
        /* In - Ranking order of processes
           in new communicator */

    MPI_Comm *new_comm)
        /* Out - New communicator shared by
           processes in same partition */
```

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

Example: Create Communicators for Process Rows

```
MPI_Comm grid_comm; /* 2-D process grid */

MPI_Comm grid_coords[2];
         /* Location of process in grid */

MPI_Comm row_comm;
                  /* Processes in same row */

MPI_Comm_split (grid_comm, grid_coords[0],
   grid_coords[1], &row_comm);
```

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

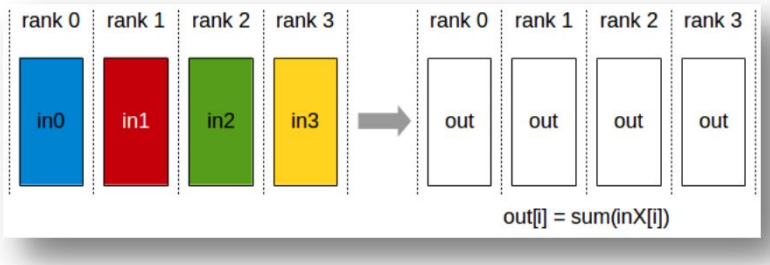
Summary

- Collective communications
  - one-to-all broadcast, all-to-one reduction
  - all-to-all broadcast, all-to-all reduction
  - gather, scatter
  - all-to-all exchange

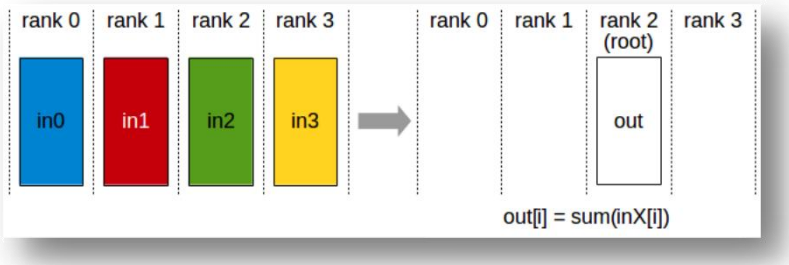
Clear Now?

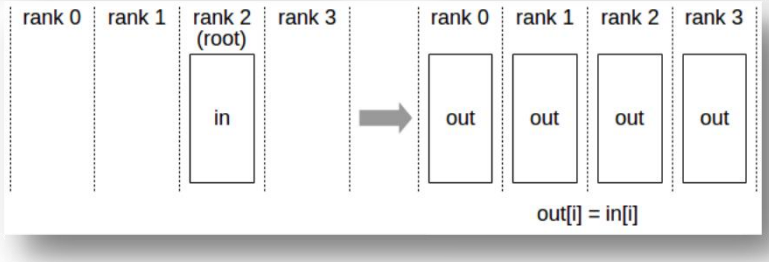
- **NVIDIA Collective Communications Library (NCCL)**

ncclAllReduce()

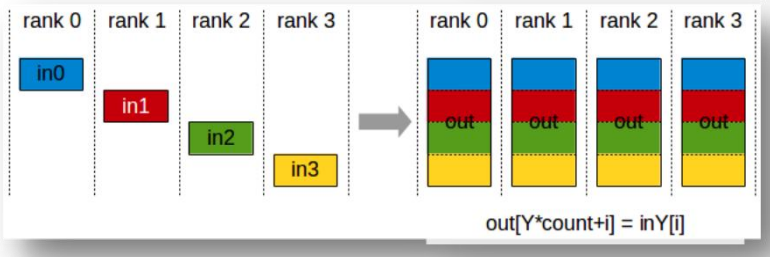


ncclReduce

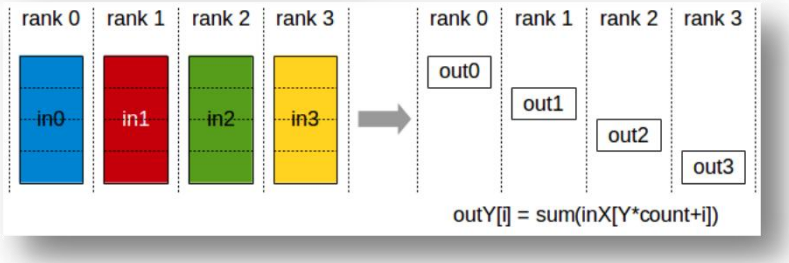




ncclBroadcast()



ncclAllGather()



ncclReduceScatter()