CSCI 5451: Introduction to Parallel Computing

Lecture 9: Basic Communication Operations

computer science
& engineering

University of Minnesota
Driven to Discover®

Announcements (9/29)

Cuda Machine → If you have been unable to access the CUDA machines, please fill out this form detailing your specific problems.

Lecture Overview

- Homework 1
- Basic Communication Operations
  - Overview
  - One-to-All Broadcast & All-to-One Reduction
  - All-to-All Broadcast & All-to-All Reduction
  - All-Reduce & Prefix-Sum
  - Scatter & Gather
  - All-to-All Personalized Communication

Lecture Overview

- **Homework 1**
- Basic Communication Operations
  - Overview
  - One-to-All Broadcast & All-to-One Reduction
  - All-to-All Broadcast & All-to-All Reduction
  - All-Reduce & Prefix-Sum
  - Scatter & Gather
  - All-to-All Personalized Communication

Dynamic Time Warping Algorithm

Algorithm 1: Dynamic Time Warping (DTW)

Input: Sequences $\mathbf{A}[0..m-1]$, $\mathbf{B}[0..n-1]$ (vectors of dimension d)
Output: Minimum alignment cost $D[m-1][n-1]$
function DTW($\mathbf{A}, \mathbf{B}$);

```
let D[0..m-1][0..n-1];
let dist[0..m-1][0..n-1];
// Set first timer after allocating these arrays

for i ← 0 to m-1 do
    for j ← 0 to n-1 do
        dist[i][j] ← ||A[i] - B[j]||_2;
    end
end

D[0][0] ← dist[0][0];
for i ← 1 to m-1 do
    D[i][0] ← dist[i][0] + D[i-1][0];
end
for j ← 1 to n-1 do
    D[0][j] ← dist[0][j] + D[0][j-1];
end

for i ← 1 to m-1 do
    for j ← 1 to n-1 do
        D[i][j] ← dist[i][j] + min(D[i-1][j], D[i][j-1], D[i-1][j-1]);
    end
end
// Set last timer after completing the above computation

return D[m-1][n-1];
```

Dynamic Time Warping Example

Algorithm 1: Dynamic Time Warping (DTW)

Input: Sequences $\mathbf{A}[0..m-1]$, $\mathbf{B}[0..n-1]$ (vectors of dimension d)
Output: Minimum alignment cost $D[m-1][n-1]$
function DTW($\mathbf{A}, \mathbf{B}$);

let $D[0..m-1][0..n-1]$;
let $\text{dist}[0..m-1][0..n-1]$;
`// Set first timer after allocating these arrays`

for $i \leftarrow 0$ ***to*** $m-1$ **do**
  for $j \leftarrow 0$ ***to*** $n-1$ **do**
    $\text{dist}[i][j] \leftarrow \|A[i] - B[j]\|_2$;
  end
end

$D[0][0] \leftarrow \text{dist}[0][0]$;
for $i \leftarrow 1$ ***to*** $m-1$ **do**
  $D[i][0] \leftarrow \text{dist}[i][0] + D[i-1][0]$;
end
for $j \leftarrow 1$ ***to*** $n-1$ **do**
  $D[0][j] \leftarrow \text{dist}[0][j] + D[0][j-1]$;
end

for $i \leftarrow 1$ ***to*** $m-1$ **do**
  for $j \leftarrow 1$ ***to*** $n-1$ **do**
    $D[i][j] \leftarrow \text{dist}[i][j] + \min(D[i-1][j], D[i][j-1], D[i-1][j-1])$;
  end
end
`// Set last timer after completing the above computation`

return $D[m-1][n-1]$;

$$A_1 = (0,0),\ A_2 = (1,1),\ A_3 = (2,2)$$
$$B_1 = (0,1),\ B_2 = (1,2),\ B_3 = (2,1),\ B_4 = (3,1)$$

Dynamic Time Warping Example

Algorithm 1: Dynamic Time Warping (DTW)

Input: Sequences $\mathbf{A}[0..m-1]$, $\mathbf{B}[0..n-1]$ (vectors of dimension d)
Output: Minimum alignment cost $D[m-1][n-1]$
function DTW($\mathbf{A}, \mathbf{B}$);

```
let D[0..m−1][0..n−1];
let dist[0..m−1][0..n−1];
// Set first timer after allocating these arrays
for i ← 0 to m−1 do
    for j ← 0 to n−1 do
        dist[i][j] ← ‖A[i] − B[j]‖₂;
    end
end
D[0][0] ← dist[0][0];
for i ← 1 to m−1 do
    D[i][0] ← dist[i][0] + D[i−1][0];
end
for j ← 1 to n−1 do
    D[0][j] ← dist[0][j] + D[0][j−1];
end
for i ← 1 to m−1 do
    for j ← 1 to n−1 do
        D[i][j] ← dist[i][j] + min(D[i−1][j], D[i][j−1], D[i−1][j−1]);
    end
end
// Set last timer after completing the above computation
return D[m−1][n−1];
```

$A_1 = (0,0),\ A_2 = (1,1),\ A_3 = (2,2)$
$B_1 = (0,1),\ B_2 = (1,2),\ B_3 = (2,1),\ B_4 = (3,1)$

	B_1	B_2	B_3	B_4
A_1	1.000	2.236	2.236	3.162
A_2	1.000	1.000	1.000	2.000
A_3	2.236	1.000	1.000	1.414

Dynamic Time Warping Example

Algorithm 1: Dynamic Time Warping (DTW)

Input: Sequences $\mathbf{A}[0..m-1]$, $\mathbf{B}[0..n-1]$ (vectors of dimension d)
Output: Minimum alignment cost $D[m-1][n-1]$
function DTW($\mathbf{A}, \mathbf{B}$);

let $D[0..m-1][0..n-1]$;
let $\text{dist}[0..m-1][0..n-1]$;
`// Set first timer after allocating these arrays`

for $i \leftarrow 0$ ***to*** $m-1$ **do**
    for $j \leftarrow 0$ ***to*** $n-1$ **do**
        $\text{dist}[i][j] \leftarrow \|A[i] - B[j]\|_2$;
    end
end
$D[0][0] \leftarrow \text{dist}[0][0]$;
for $i \leftarrow 1$ ***to*** $m-1$ **do**
    $D[i][0] \leftarrow \text{dist}[i][0] + D[i-1][0]$;
end
for $j \leftarrow 1$ ***to*** $n-1$ **do**
    $D[0][j] \leftarrow \text{dist}[0][j] + D[0][j-1]$;
end
for $i \leftarrow 1$ ***to*** $m-1$ **do**
    for $j \leftarrow 1$ ***to*** $n-1$ **do**
        $D[i][j] \leftarrow \text{dist}[i][j] + \min(D[i-1][j], D[i][j-1], D[i-1][j-1])$;
    end
end
`// Set last timer after completing the above computation`
return $D[m-1][n-1]$;

$$A_1 = (0,0),\ A_2 = (1,1),\ A_3 = (2,2)$$
$$B_1 = (0,1),\ B_2 = (1,2),\ B_3 = (2,1),\ B_4 = (3,1)$$

	B_1	B_2	B_3	B_4
A_1	1.000	2.236	2.236	3.162
A_2	1.000	1.000	1.000	2.000
A_3	2.236	1.000	1.000	1.414

	B_1	B_2	B_3	B_4
A_1	1.000	3.236	5.472	8.634
A_2	2.000	2.000	3.000	5.000
A_3	4.236	3.000	3.000	4.414

Dynamic Time Warping Example

Algorithm 1: Dynamic Time Warping (DTW)

Input: Sequences $\mathbf{A}[0..m-1]$, $\mathbf{B}[0..n-1]$ (vectors of dimension d)
Output: Minimum alignment cost $D[m-1][n-1]$
function DTW($\mathbf{A}, \mathbf{B}$);

let $D[0..m-1][0..n-1]$;
let $\text{dist}[0..m-1][0..n-1]$;
`// Set first timer after allocating these arrays`

for $i \leftarrow 0$ ***to*** $m-1$ **do**
  for $j \leftarrow 0$ ***to*** $n-1$ **do**
    $\text{dist}[i][j] \leftarrow \|A[i] - B[j]\|_2$;
  end
end

$D[0][0] \leftarrow \text{dist}[0][0]$;
for $i \leftarrow 1$ ***to*** $m-1$ **do**
  $D[i][0] \leftarrow \text{dist}[i][0] + D[i-1][0]$;
end
for $j \leftarrow 1$ ***to*** $n-1$ **do**
  $D[0][j] \leftarrow \text{dist}[0][j] + D[0][j-1]$;
end

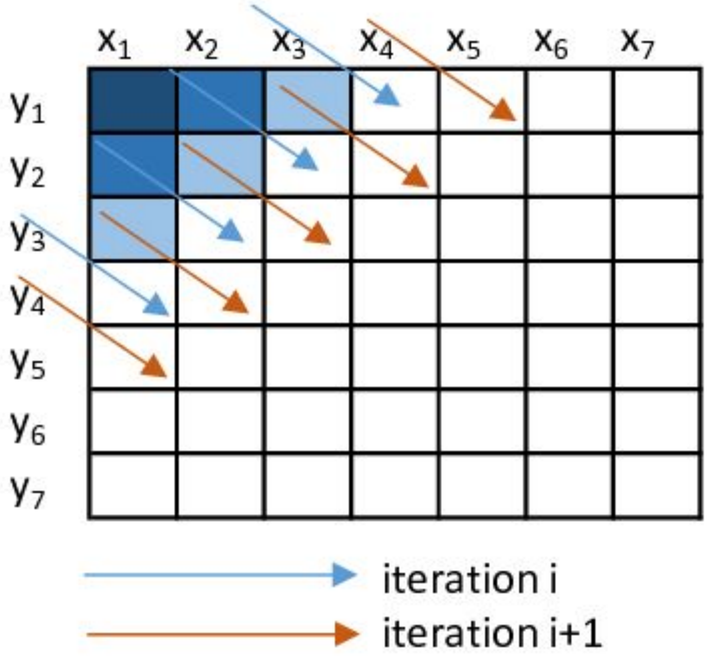
for $i \leftarrow 1$ ***to*** $m-1$ **do**
  for $j \leftarrow 1$ ***to*** $n-1$ **do**
    $D[i][j] \leftarrow \text{dist}[i][j] + \min(D[i-1][j], D[i][j-1], D[i-1][j-1])$;
  end
end
`// Set last timer after completing the above computation`
return $D[m-1][n-1]$;

Minimum Alignment Cost

	B_1	B_2	B_3	B_4
A_1	1.000	3.236	5.472	8.634
A_2	2.000	2.000	3.000	5.000
A_3	4.236	3.000	3.000	$\boxed{4.414}$

Wavefront Parallelism

- In order to compute ***D***, you will have to use wavefront parallelism
- The computation of this for-loop has dependencies - you must parallelize along the anti-diagonal
- In other words, you will not be able to just add OpenMP directives to the second for-loop



Submission

- dtw_parallel.c
- PDF report containing speedups on one of the tests as well as a description of your parallelization strategies

Grading Criteria

- Compiles + runs + correct + no-memory leaks
- Speedups → Your program must achieve the desired speedups using OpenMP as instructed
- Report → Contains speedup figure + describes parallelization strategy

Be sure to review all the tips

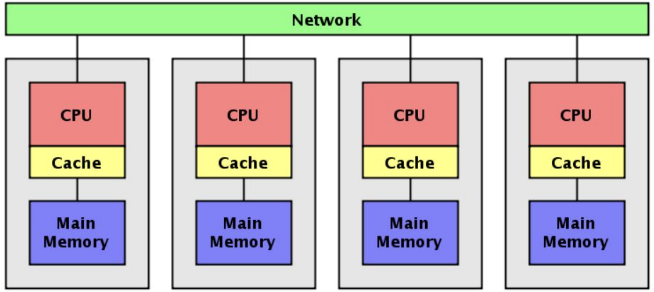
If you have questions, reach out in the homework slack channel #hw1

Lecture Overview

- Homework 1
- **Basic Communication Operations**
  - **Overview**
  - One-to-All Broadcast & All-to-One Reduction
  - All-to-All Broadcast & All-to-All Reduction
  - All-Reduce & Prefix-Sum
  - Scatter & Gather
  - All-to-All Personalized Communication

Communication Operations (Distributed Memory)

- The last few lectures we covered shared-memory programs
- We will now return to distributed memory programs
- Specifically, we will be looking at how we can get multiple processors to communicate concurrently



Source: Kaminsky/Parallel Java

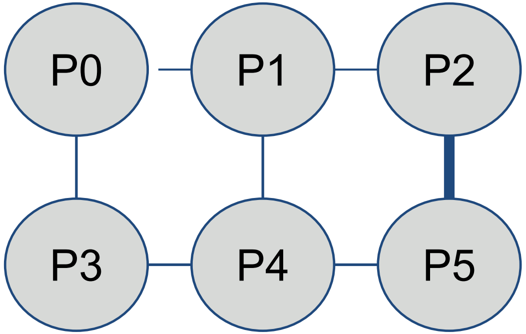
General-Purpose Interactions

- In most distributed programs with communication - communications will often occur between *all* or some *large subset* of processors in a given topology (very rarely will there be single processes which communicate)
- We will define some of these more common communication patterns in today's lecture & discuss how to efficiently perform them on certain physical processor topologies

General-Purpose Interactions

- In most distributed programs with communication - communications will often occur between *all* or some *large subset* of processors in a given topology (very rarely will there be single processes which communicate)
- We will define some of these more common communication patterns in today's lecture & discuss how to efficiently perform them on certain physical processor topologies

Example of Unlikely Communication: P2 sends a message to P5 while no other communications occur

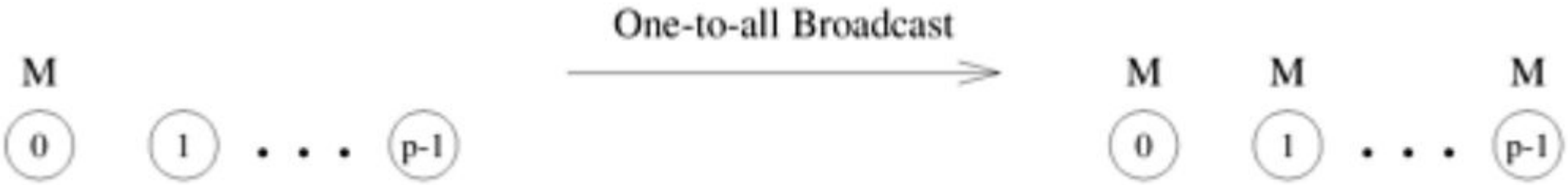


Lecture Overview

- Homework 1
- **Basic Communication Operations**
  - Overview
  - **One-to-All Broadcast & All-to-One Reduction**
  - All-to-All Broadcast & All-to-All Reduction
  - All-Reduce & Prefix-Sum
  - Scatter & Gather
  - All-to-All Personalized Communication

One-to All Broadcast

A message ***M*** exists on one processor which we want to send to all *p* other processors



How can we efficiently get this message across all *p* processors?

How can we efficiently get this message across all *p* processors?

Use Recursive Doubling (examples on following slides):

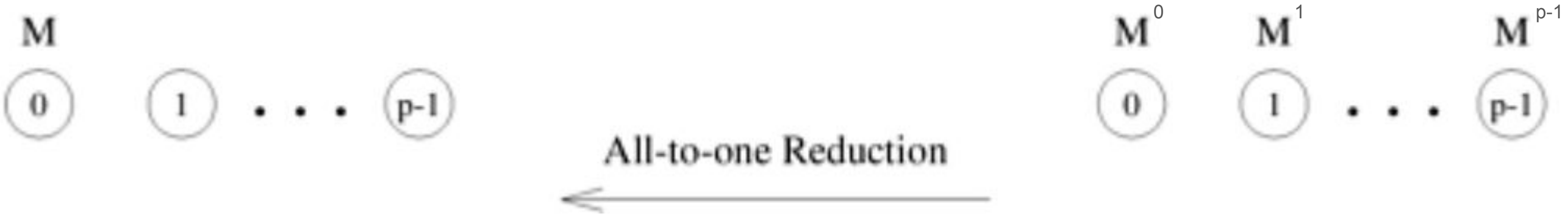
1. **P**$_0$ sends to **P**$_{p/2}$
2. [In parallel] **P**$_0$ sends to **P**$_{p/4}$ |*and*| **P**$_{p/2}$ sends to **P**$_{3p/4}$

…

log(p). [In parallel] **P**$_0$ sends to **P**$_1$ |and| **P**$_2$ sends to **P**$_3$ |and| … |and| **P**$_{p-2}$ sends to **P**$_{p-1}$

All-to-One Reduction

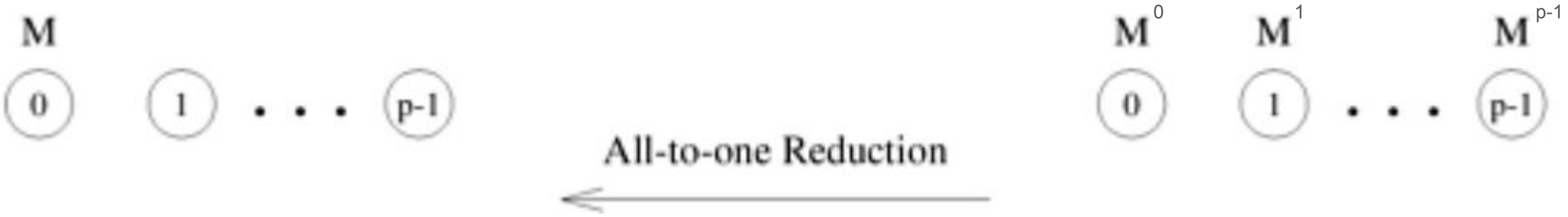
Messages $\boldsymbol{M^i}$ exist on each process *i*,
and we want to combine these
messages onto a single processor.



All-to-One Reduction

The operation we use to combine messages can be sums, products, maximums, minimums, etc.

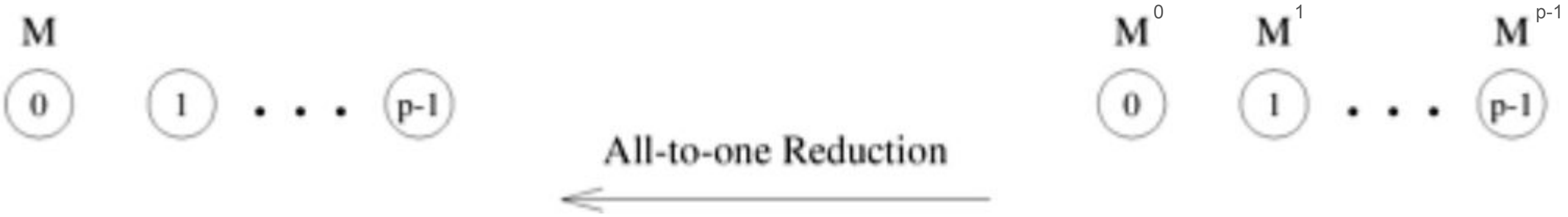
Messages M^i exist on each process i, and we want to combine these messages onto a single processor.



All-to-One Reduction

For example, if we combined the below messages with 'sum', then we would have $\boldsymbol{M = M^0 + M^1 + \ldots + M^{p-1}}$

Messages $\boldsymbol{M^i}$ exist on each process *i*, and we want to combine these messages onto a single processor.



All-to-One Reduction <> One-to-All Broadcast

All-to-One Reduction is the dual of One-to-All Broadcast. We can think of All-to-One Reduction as running in the operation direction as One-to-All Broadcast

All-to-One Reduction <> One-to-All Broadcast

All-to-One Reduction is the dual of One-to-All Broadcast. We can think of All-to-One Reduction as running in the operation direction as One-to-All Broadcast

Use Recursive Halving (examples on following slides):

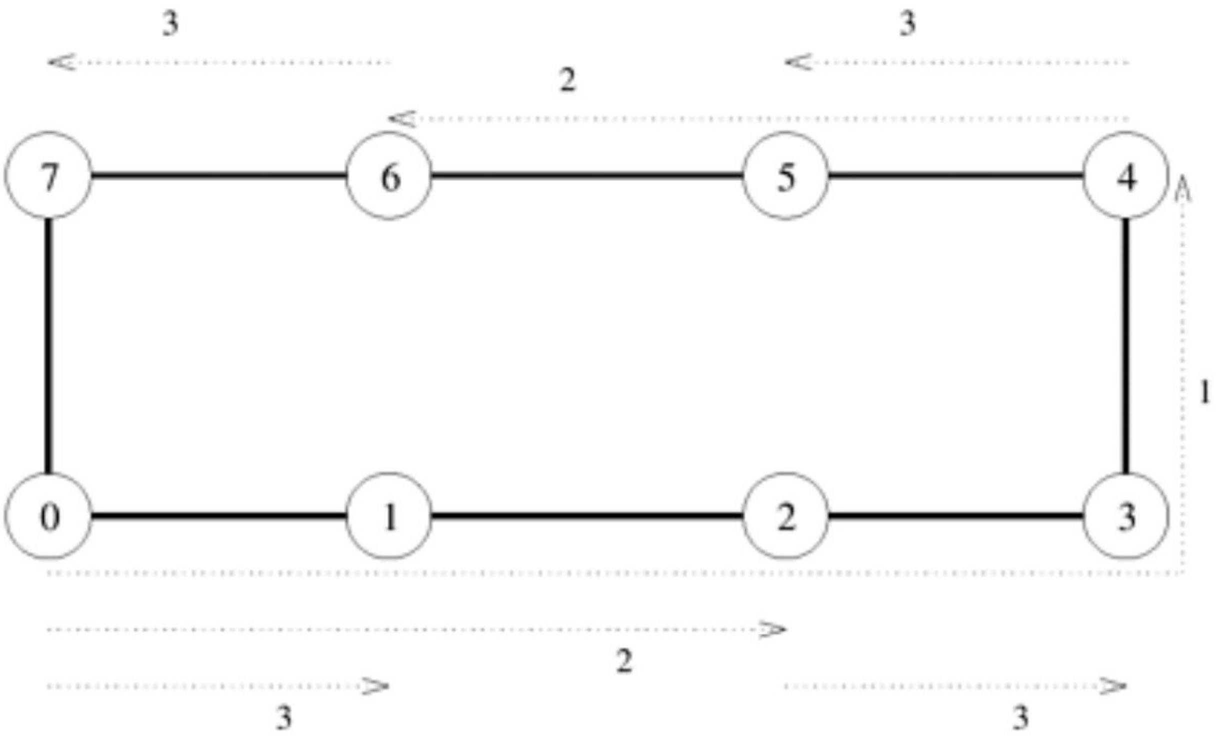
1. [In parallel] (**P_1** sends to **P_0** and **P_0** sums) |and| (**P_3** sends to **P_2** and **P_2** sums) |and| … |and| (**P_{p-1}** sends to **P_{p-2}** and **P_{p-2}** sums)

…

log(p). (**$P_{p/2}$** sends to **P_0** and **P_0** sums)

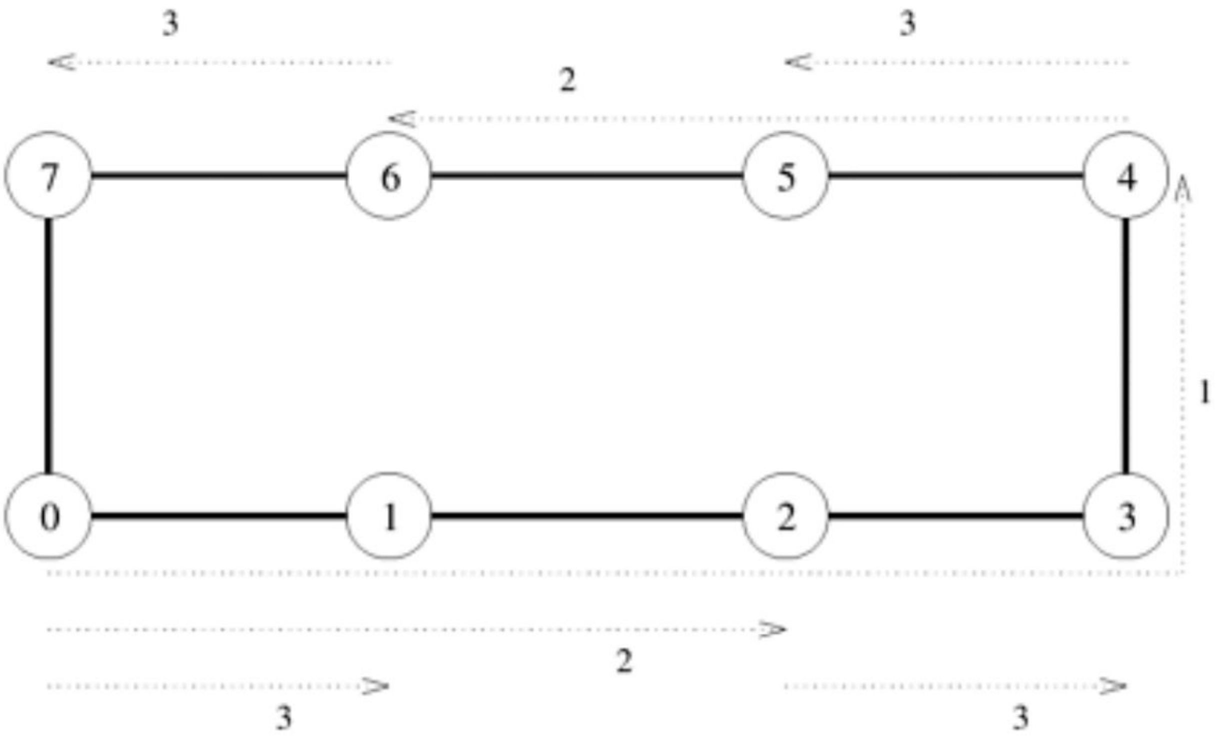
How do we map this pattern onto physical processor topologies (Ring, Mesh, Hypercube, etc.)?

One-to-All Broadcast (Ring)

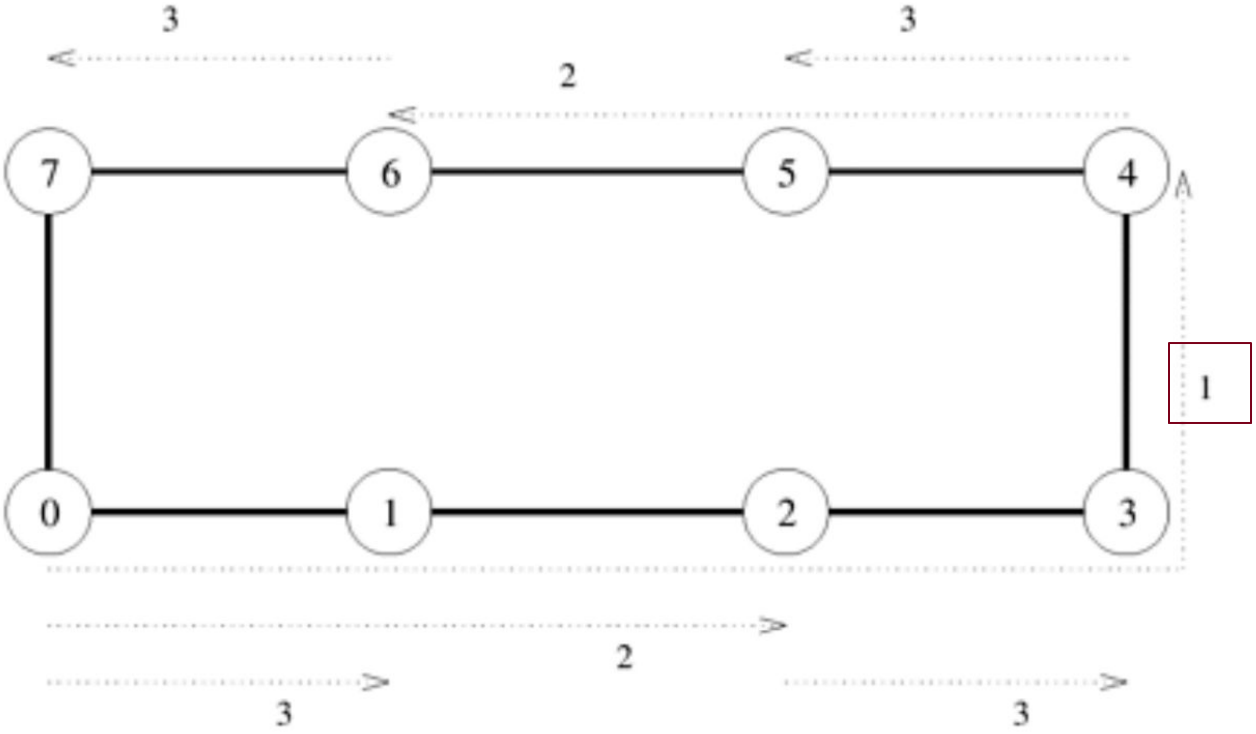


One-to-All Broadcast (Ring)

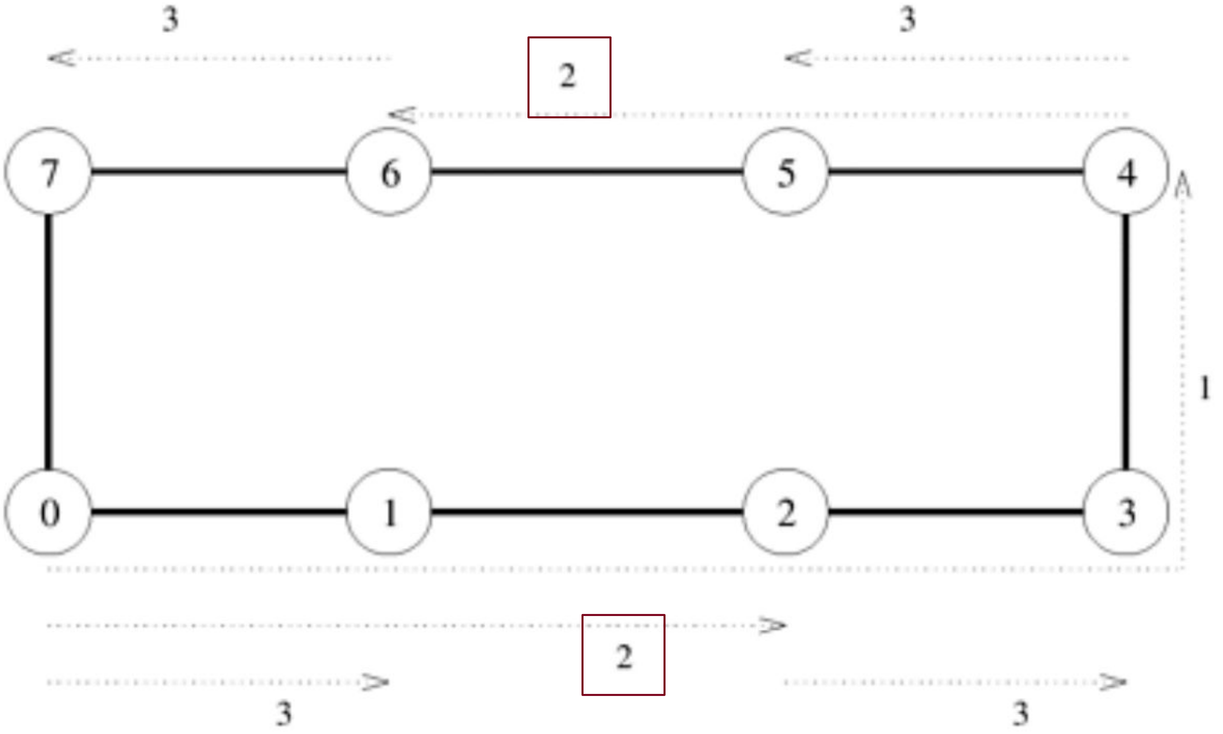
No contention on any links
at any communication step



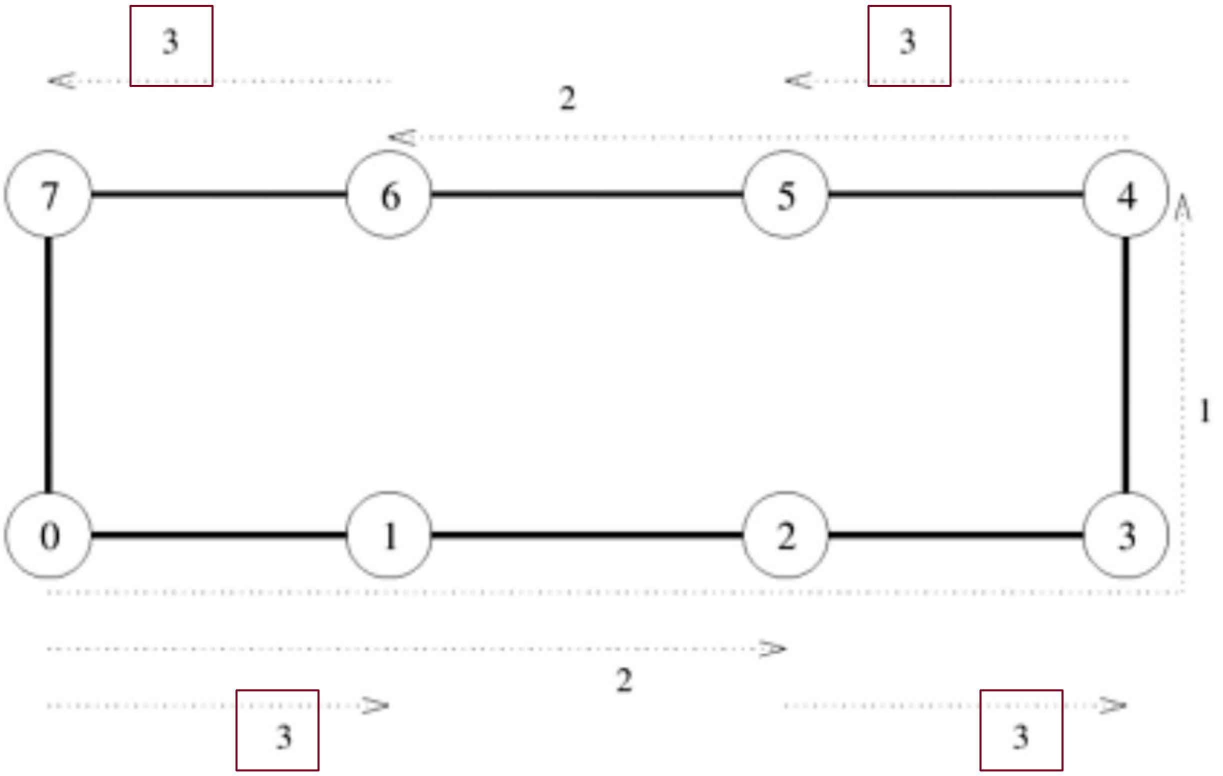
One-to-All Broadcast (Ring)



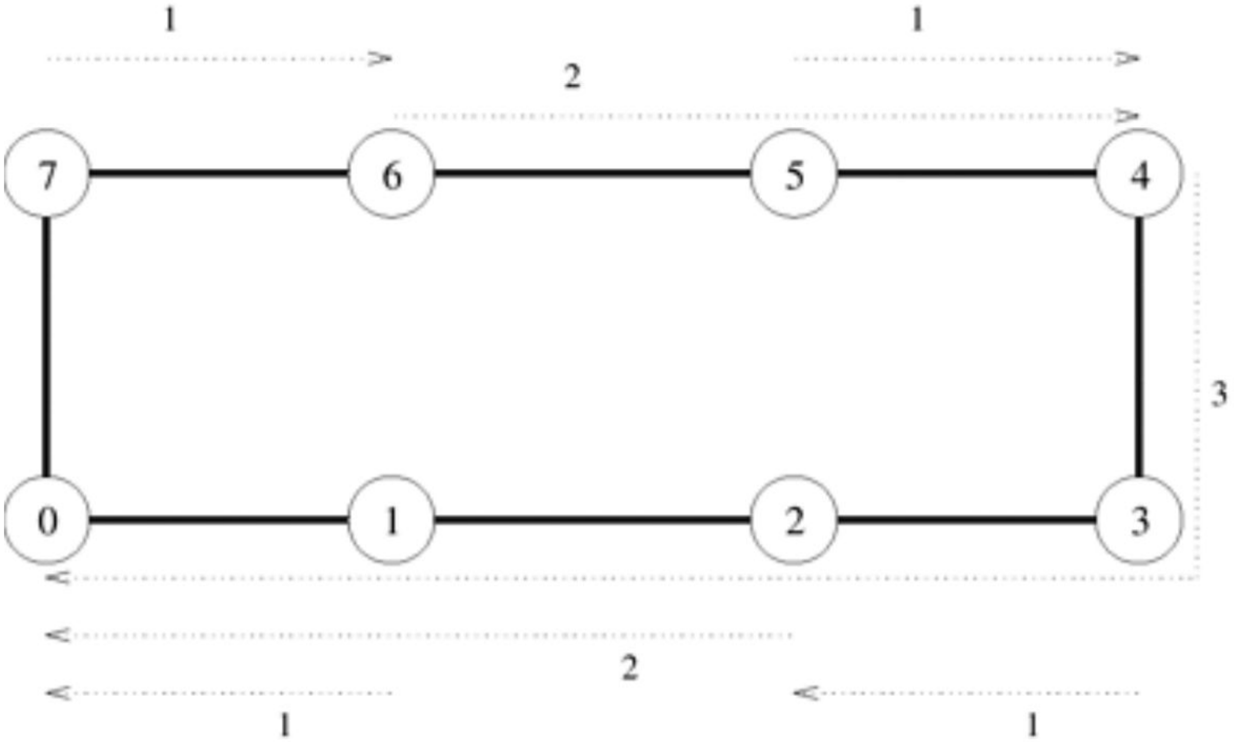
One-to-All Broadcast (Ring)



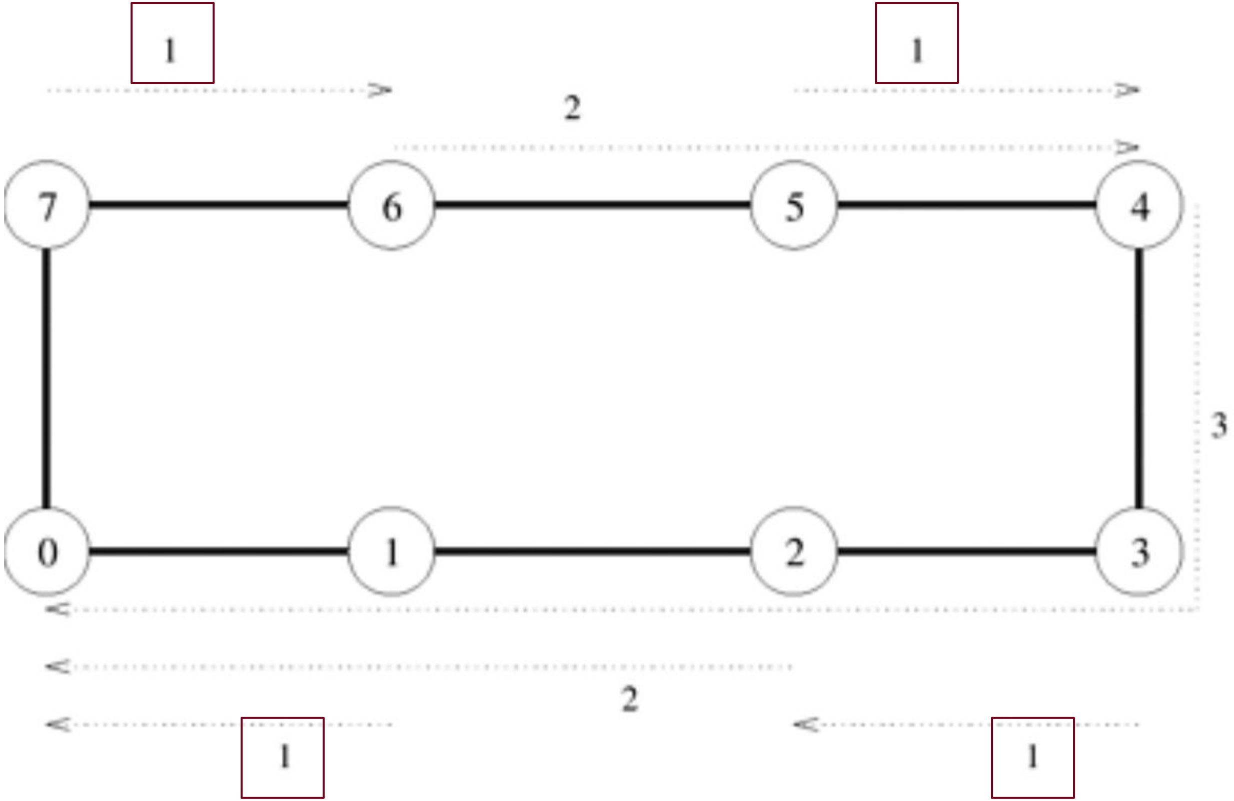
One-to-All Broadcast (Ring)



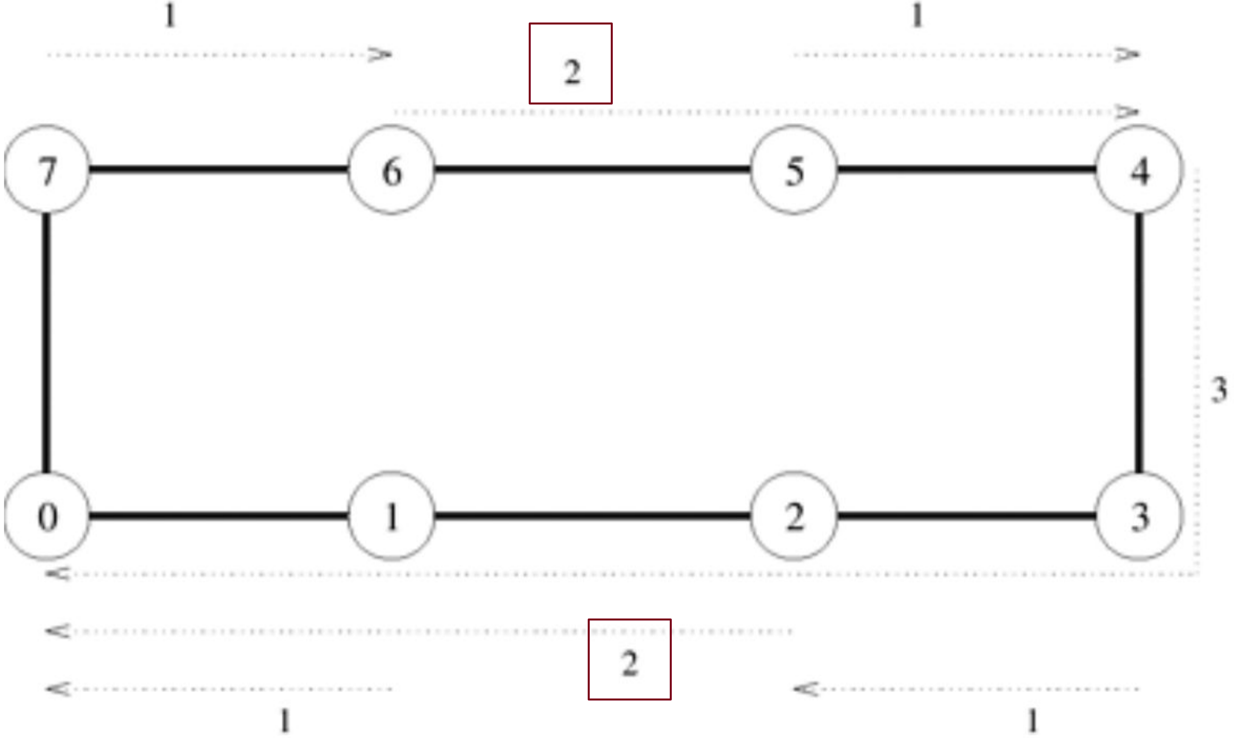
All-to-One Reduction (Ring)



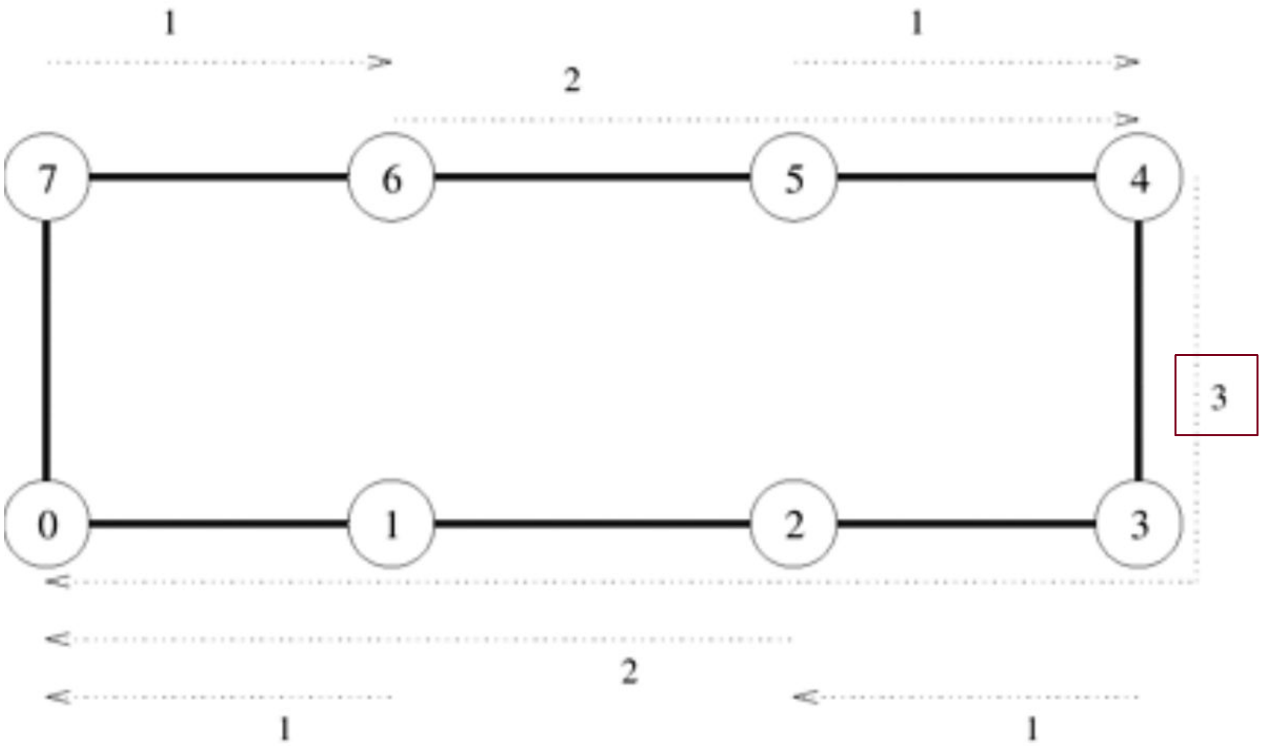
All-to-One Reduction (Ring)



All-to-One Reduction (Ring)

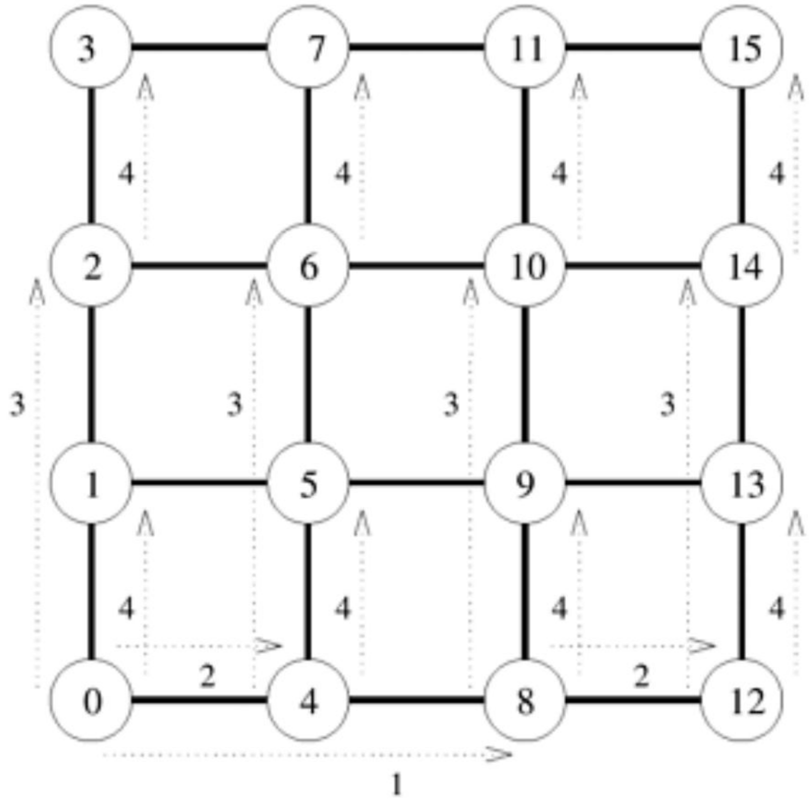


All-to-One Reduction (Ring)



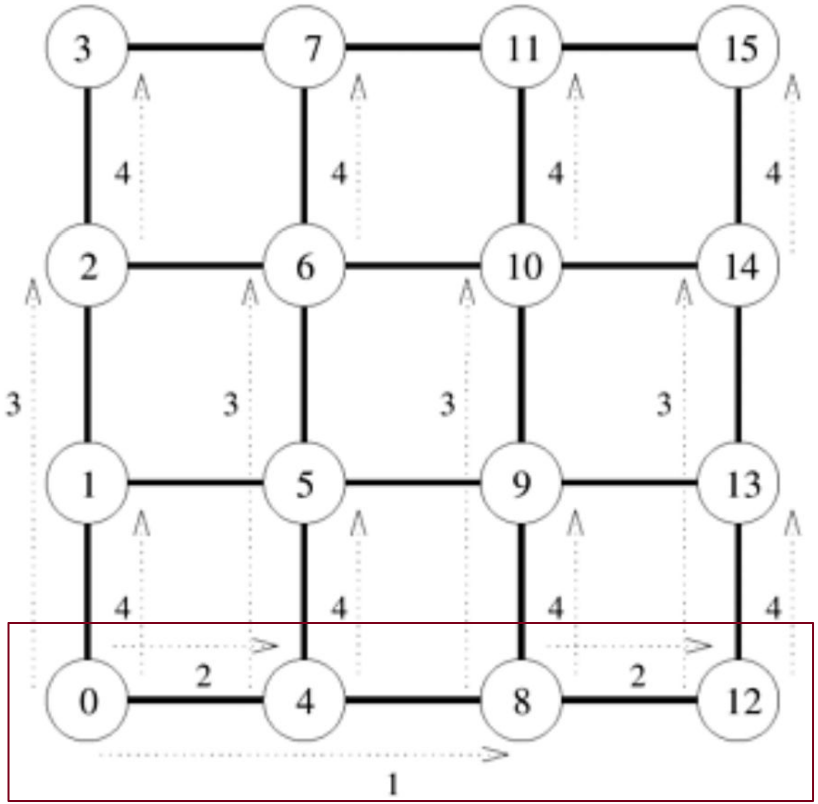
One-to-All Broadcast (Mesh)

We can treat the 2-D Mesh as mutually exclusive sets of Rings, then use our previous recursive doubling pattern



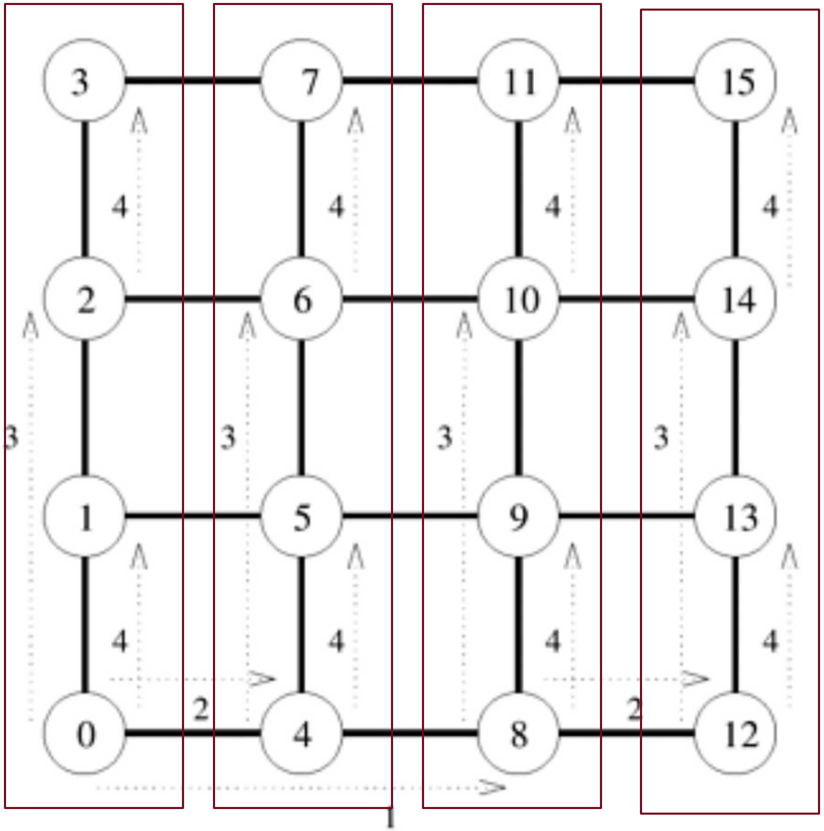
One-to-All Broadcast (Mesh)

We can treat the 2-D Mesh as mutually exclusive sets of Rings, then use our previous recursive doubling pattern



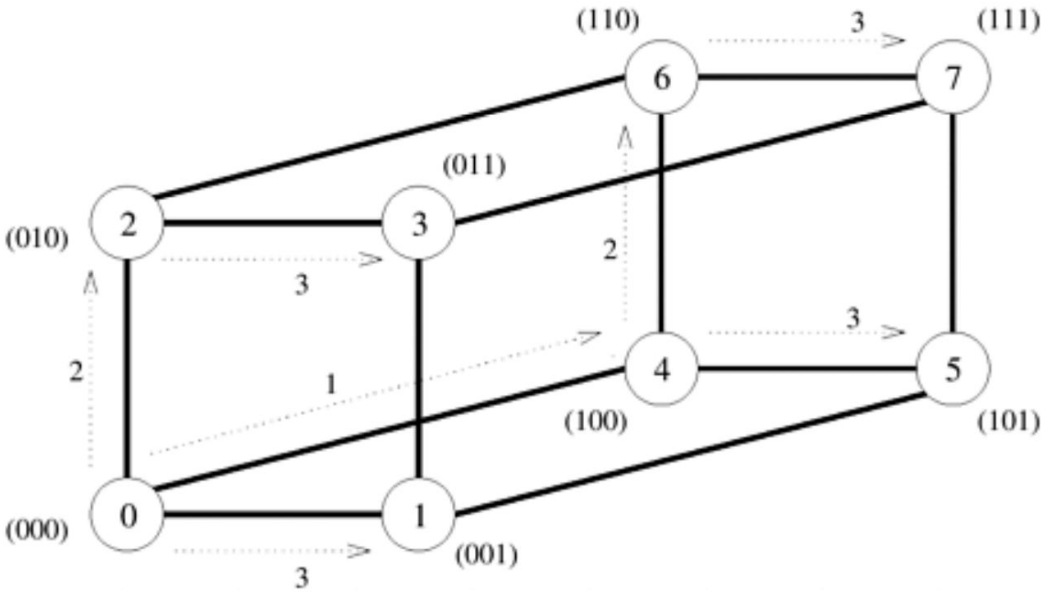
One-to-All Broadcast (Mesh)

We can treat the 2-D Mesh as mutually exclusive sets of Rings, then use our previous recursive doubling pattern



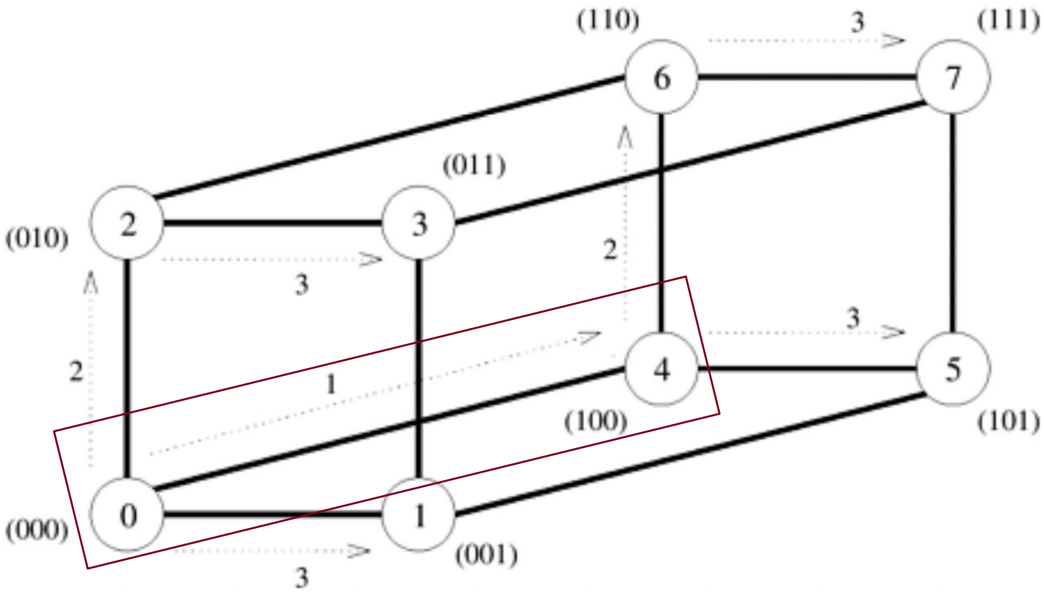
One-to-All Broadcast (Hypercube)

We can treat each bit dimension as sets of 2-process Rings. We communicate starting from the most significant bits to the least



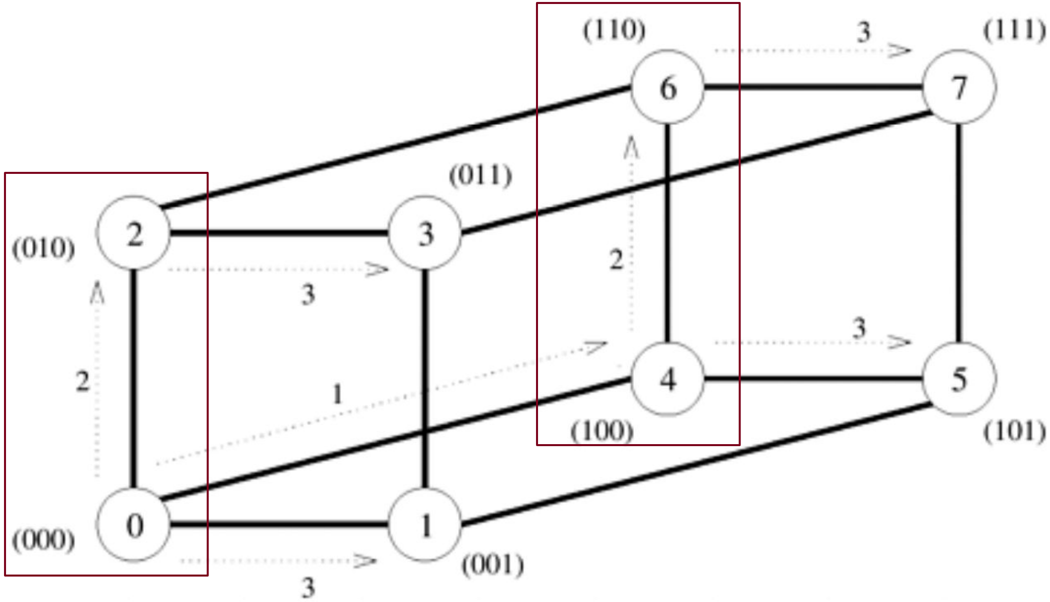
One-to-All Broadcast (Hypercube)

We can treat each bit dimension as sets of 2-process Rings. We communicate starting from the most significant bits to the least



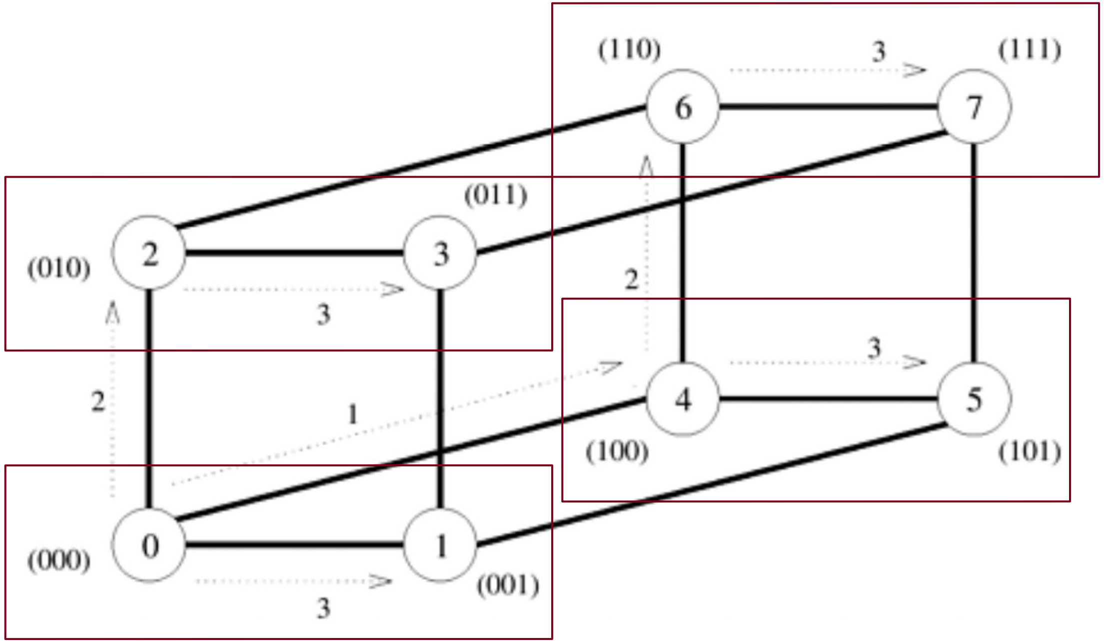
One-to-All Broadcast (Hypercube)

We can treat each bit dimension as sets of 2-process Rings. We communicate starting from the most significant bits to the least



One-to-All Broadcast (Hypercube)

We can treat each bit dimension as sets of 2-process Rings. We communicate starting from the most significant bits to the least



Cost (Time to Communicate)

- Either communication (One-to-All Reduction or All-to-One Broadcast) requires log*(p)* steps
- Each communication involves a message of size *m* (recall from our previously lecture this takes $t_s + t_w m$)
- NOTE: We omit the 'per-hop' travel times from our equation as they are typically quite small.

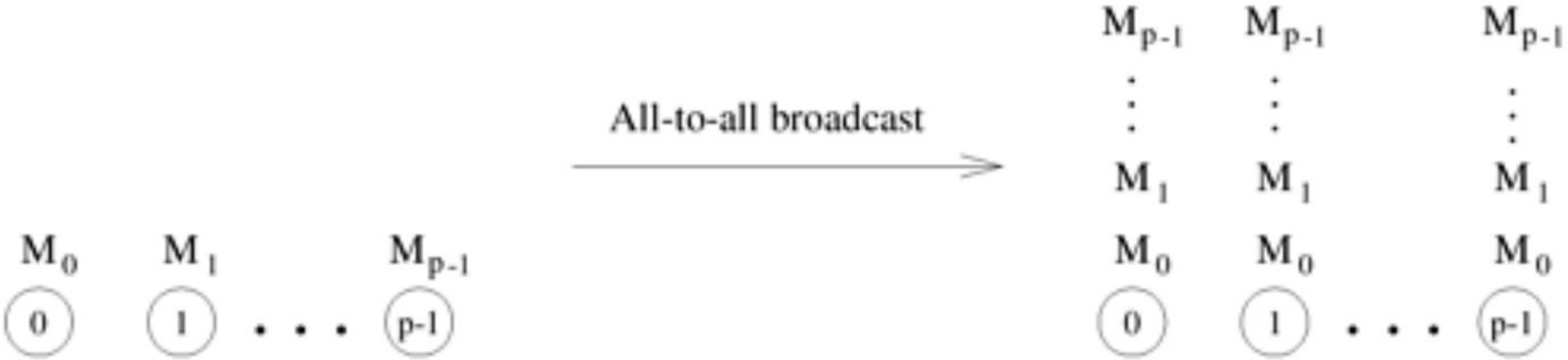
$$T = (t_s + t_w m)\log p$$

Lecture Overview

- Homework 1
- **Basic Communication Operations**
  - Overview
  - One-to-All Broadcast & All-to-One Reduction
  - **All-to-All Broadcast & All-to-All Reduction**
  - All-Reduce & Prefix-Sum
  - Scatter & Gather
  - All-to-All Personalized Communication

All-to-All Broadcast

Messages M^i exist on each process *i*, and we want each message to be present on all processors.



All-to-All Reduction

The dual of All-to-All Broadcast. Each processor has *p* messages. We want to reduce each of these onto each processor, separately

All-to-All Reduction

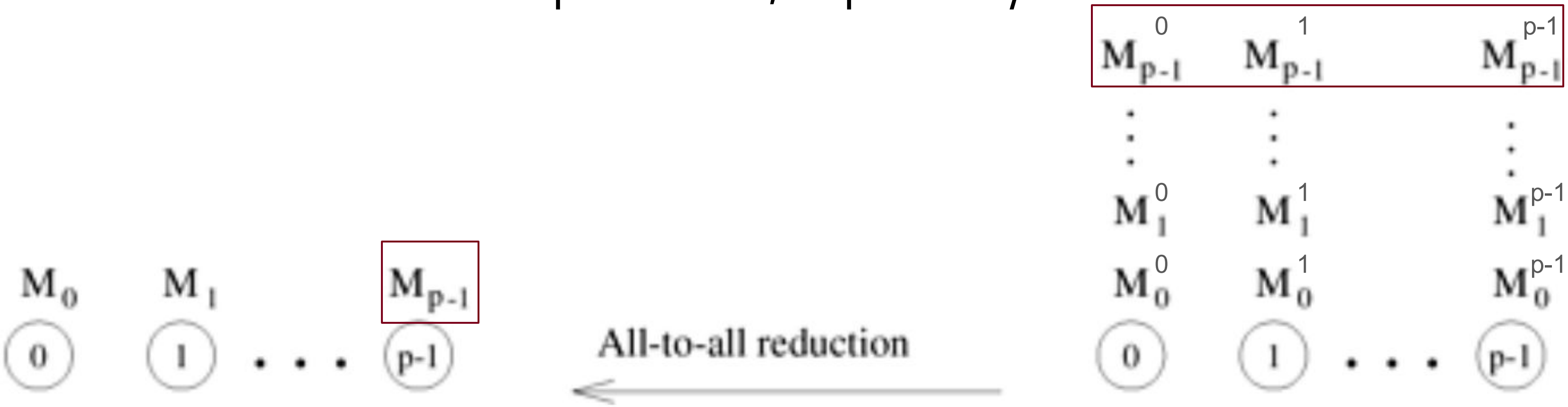
The dual of All-to-All Broadcast. Each processor has *p* messages. We want to reduce each of these onto each processor, separately

All-to-All Reduction

The dual of All-to-All Broadcast. Each processor has *p* messages. We want to reduce each of these onto each processor, separately

All-to-All Reduction

The dual of All-to-All Broadcast. Each processor has *p* messages. We want to reduce each of these onto each processor, separately



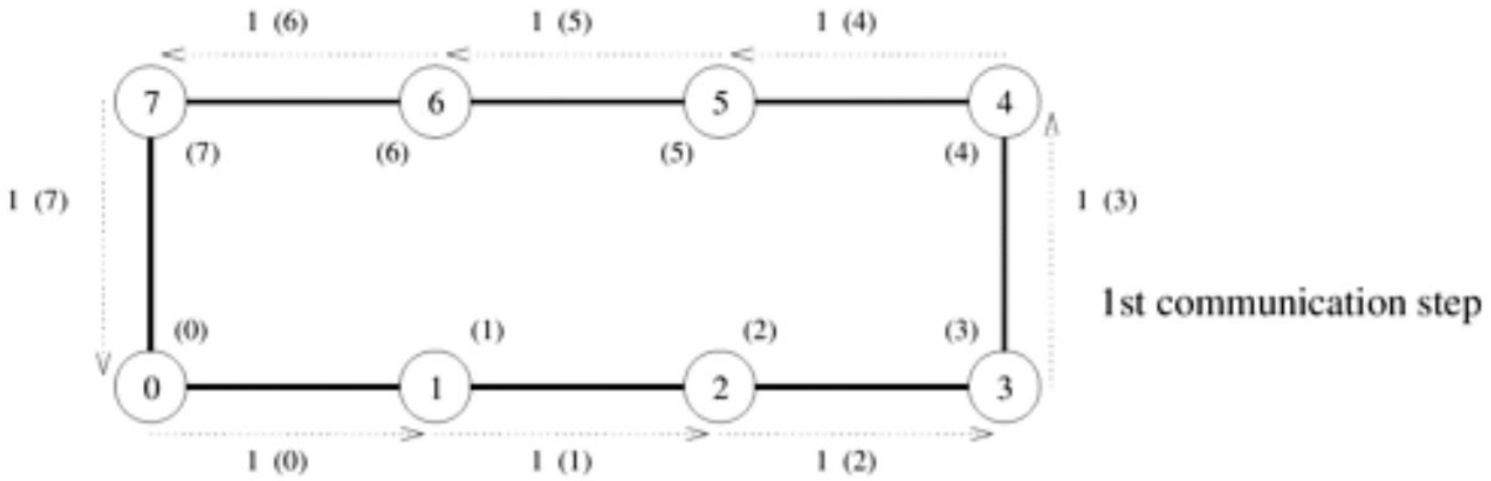
How do we map All-to-All Broadcast onto physical processor topologies (Ring, Mesh, Hypercube, etc.)? Should we just perform *p* separate One-to-All Broadcasts?

How do we map All-to-All Broadcast onto physical processor topologies (Ring, Mesh, Hypercube, etc.)? Should we just perform *p* separate One-to-All Broadcasts?

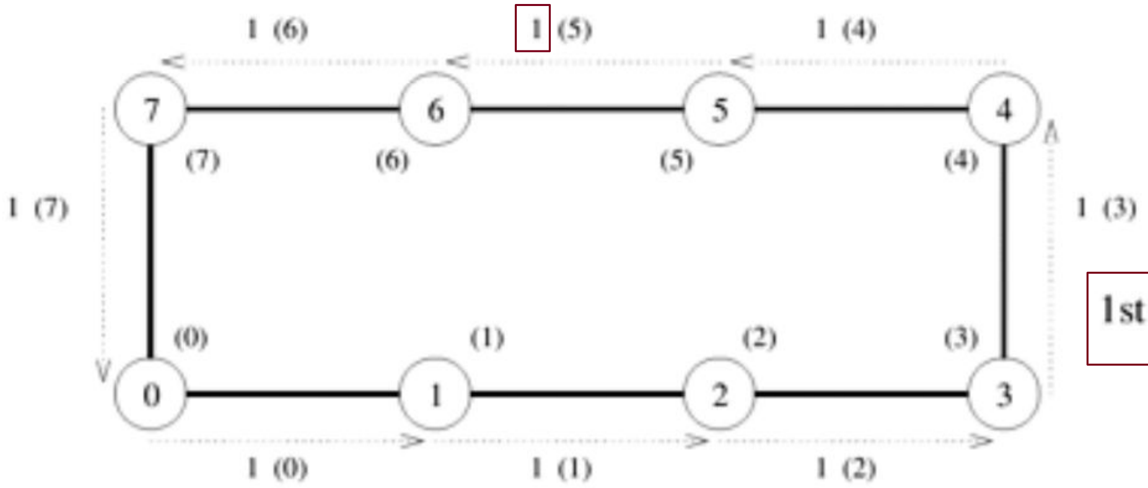
No. This will leave many links idle. We can instead have all links communicate on each step for greater speedups.

All-to-All Broadcast (Ring)

General pattern → Pass along the last message received to the next processor. This way all links are used at each step of communication.



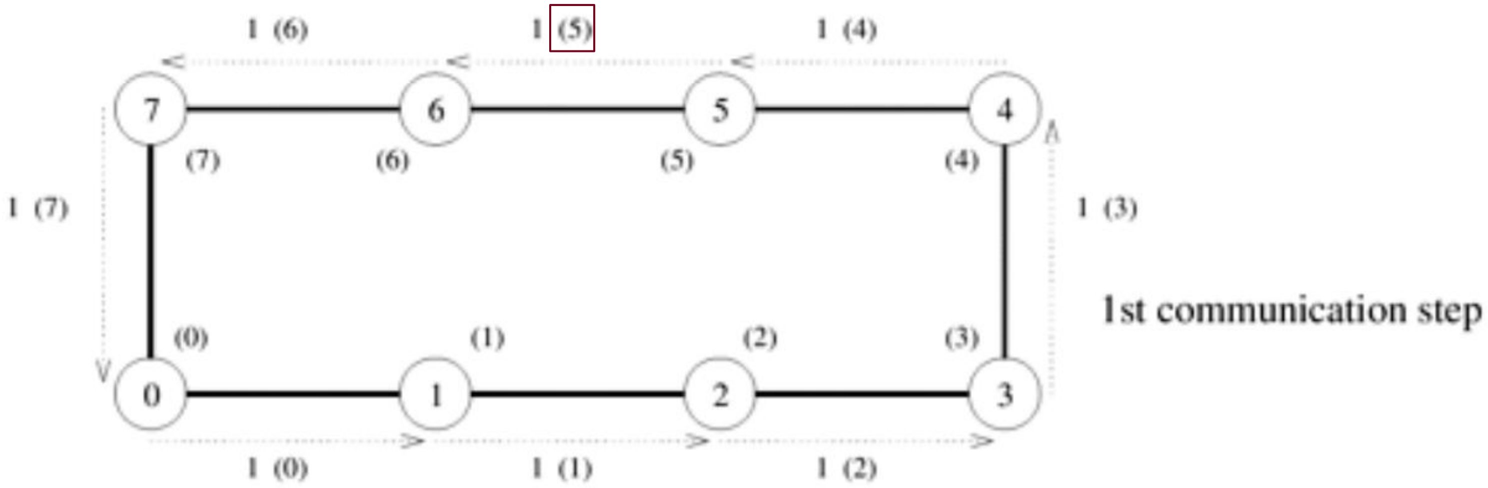
All-to-All Broadcast (Ring)



1st communication step

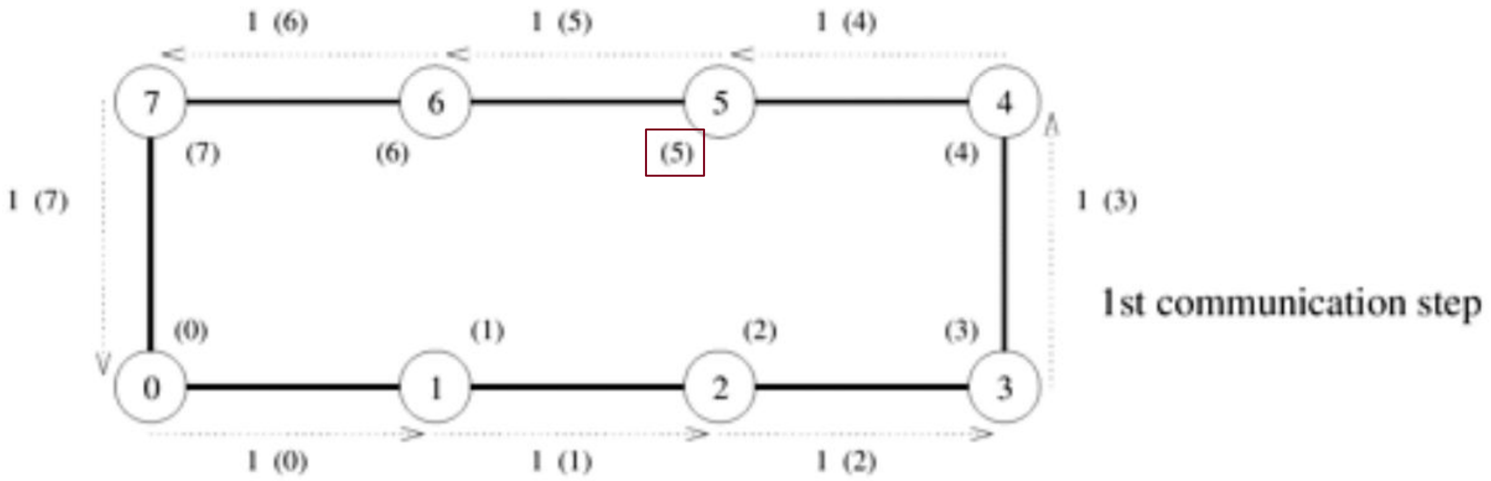
All-to-All Broadcast (Ring)

Message sent between processors

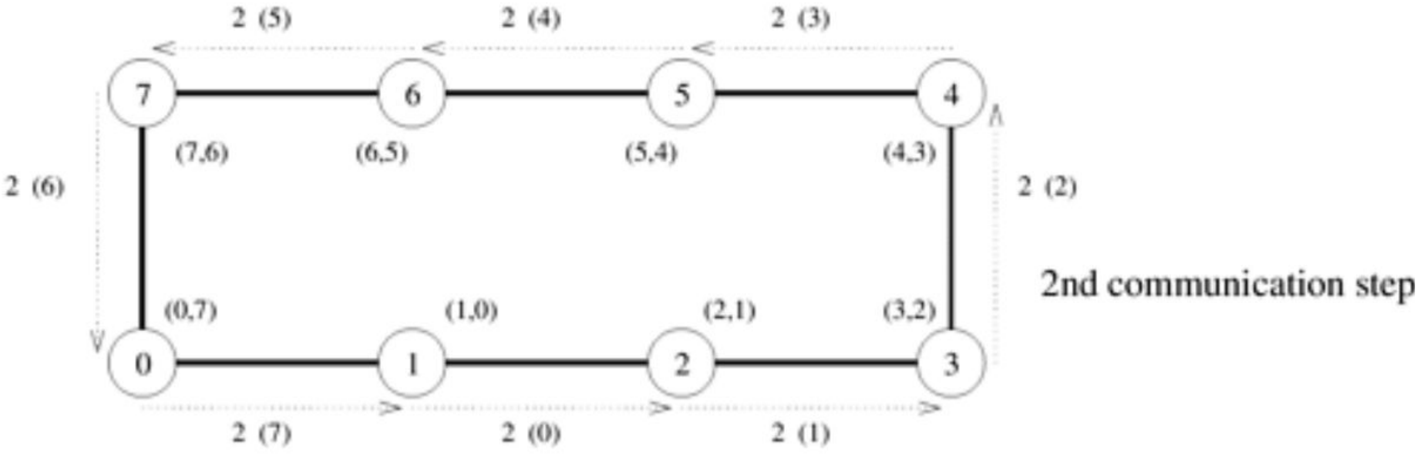


All-to-All Broadcast (Ring)

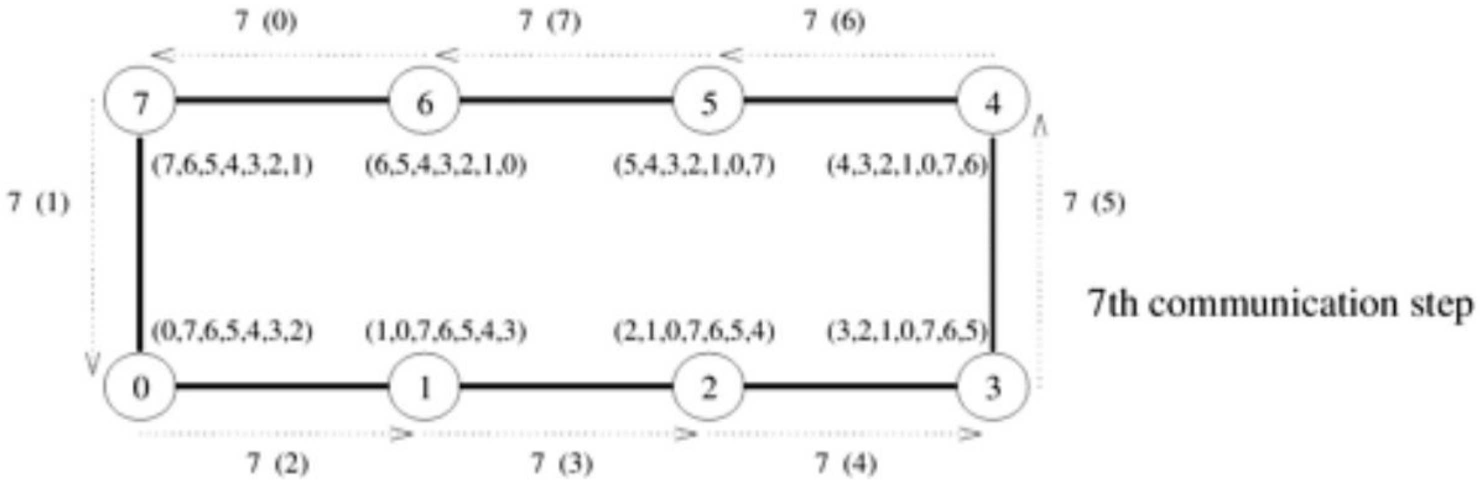
Messages currently on given processor



All-to-All Broadcast (Ring)



All-to-All Broadcast (Ring)



All-to-All Broadcast Cost (Ring)

$$T = \underbrace{\boxed{(t_s + t_w m)}}_{\text{Time per step}}\underbrace{\boxed{(p - 1)}}_{\text{Total Steps}}$$

What is the fastest way of performing All-to-All Reduction on a Ring?

What is the fastest way of performing All-to-All Reduction on a Ring?

It depends. The time using the inverse approach of All-to-All Broadcast described on the previous slides takes

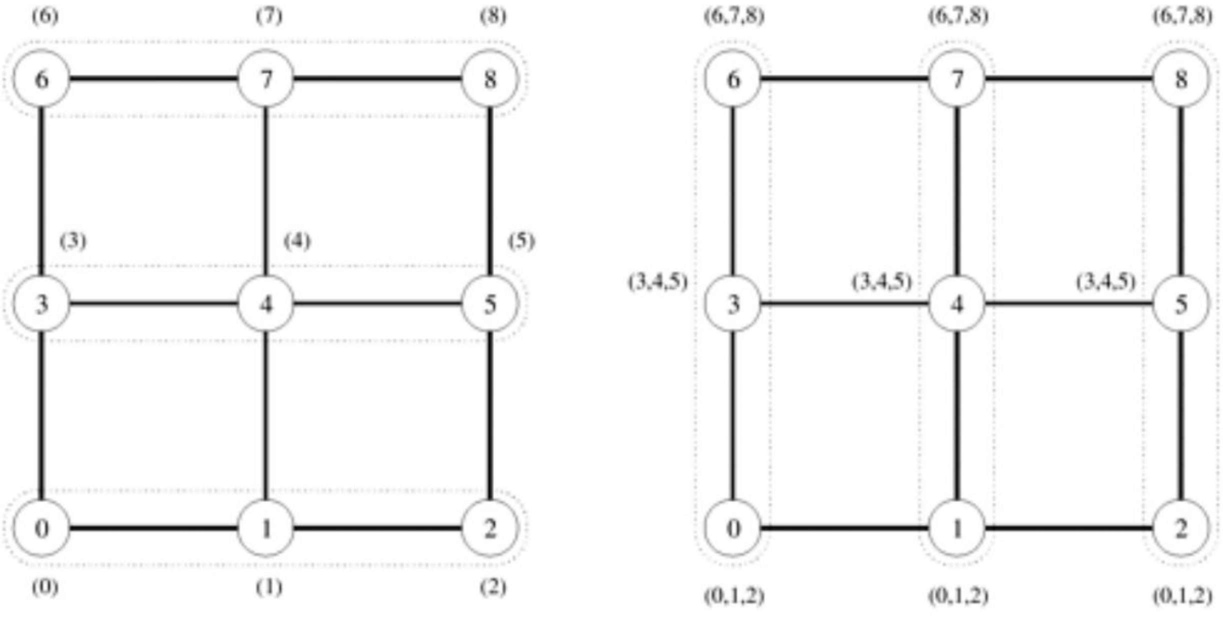
$$(p-1)(t_s + (t_w mp)/2)$$

while simply performing *p* consecutive All-to-One Reductions takes

$$p \log p(t_s + t_w m)$$

All-to-All Broadcast (Mesh)

Similar to One-to-All Broadcast, treat each dimension as a Ring



(a) Initial data distribution

(b) Data distribution after rowwise broadcast

All-to-All Broadcast Cost (Mesh)

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

All-to-All Broadcast Cost (Mesh)

$$T = \underbrace{\boxed{(t_s + t_w m)}}_{\text{Time per step on first dimension}}\underbrace{\boxed{(\sqrt{p} - 1)}}_{\text{Total Steps on first dimension}} + (t_s + t_w m\sqrt{p})(\sqrt{p} - 1)$$

All-to-All Broadcast Cost (Mesh)

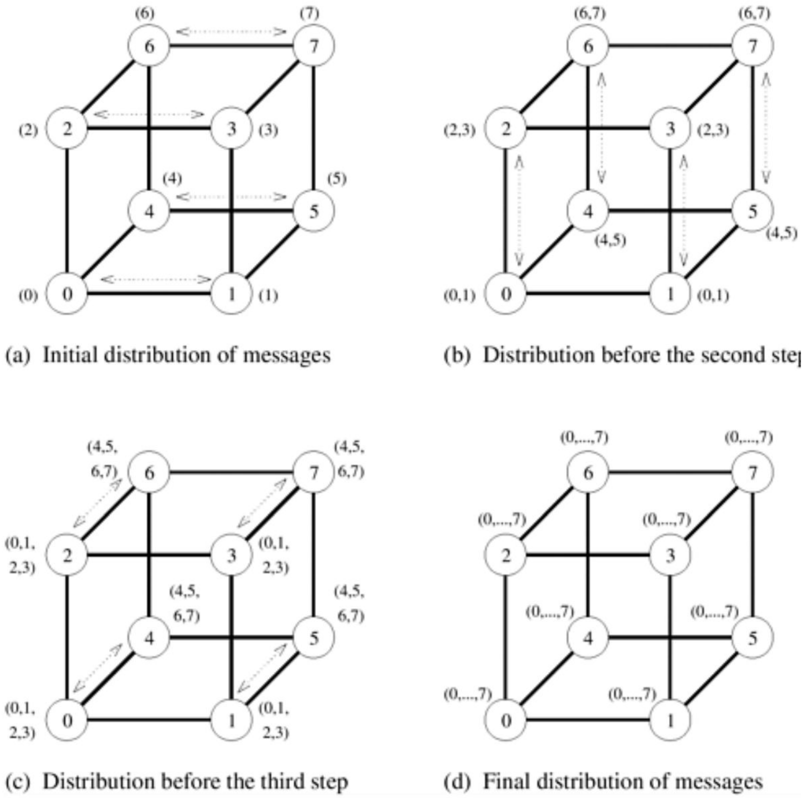
$$T = (t_s + t_w m)(\sqrt{p} - 1) + \underbrace{\boxed{(t_s + t_w m\sqrt{p})}}_{\text{Time per step on second dimension}}\underbrace{\boxed{(\sqrt{p} - 1)}}_{\text{Total Steps on second dimension}}$$

All-to-All Broadcast Cost (Mesh)

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

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

All-to-All Broadcast (Hypercube)



All-to-All Broadcast Cost (Hypercube)

$$T = (t_s + t_w m) + (t_s + 2t_w m) + \ldots + (t_s + (p/2)t_w m)$$

All-to-All Broadcast Cost (Hypercube)

$$T = \boxed{(t_s + t_w m)} + (t_s + 2t_w m) + \ldots + (t_s + (p/2)t_w m)$$

Time on first Step

All-to-All Broadcast Cost (Hypercube)

$$T = (t_s + t_w m) + \boxed{(t_s + 2t_w m)} + \ldots + (t_s + (p/2)t_w m)$$

Time on second Step

All-to-All Broadcast Cost (Hypercube)

$$T = (t_s + t_w m) + (t_s + 2t_w m) + \ldots + \boxed{(t_s + (p/2)t_w m)}$$

Time on final Step

All-to-All Broadcast Cost (Hypercube)

$$T = (t_s + t_w m) + (t_s + 2t_w m) + \ldots + (t_s + (p/2)t_w m)$$

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

Topology Matters

- The algorithm used was different in each case when changing the processor topology → How we structure communication is dependent upon the physical processor topology
- The time taken in each case was different as a result
- More links usually means faster communication

$$\boxed{T = (t_s + t_w m)(p - 1)}$$

$$\downarrow$$

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

$$\downarrow$$

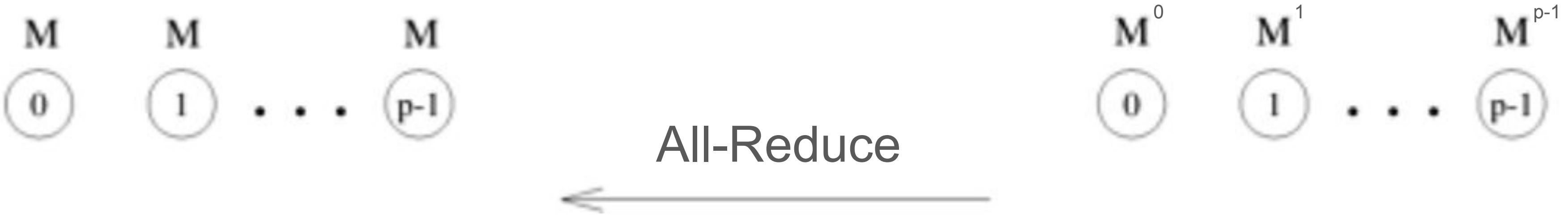
$$\boxed{T = t_s \log p + t_w m(p - 1)}$$

Lecture Overview

- Homework 1
- **Basic Communication Operations**
    - Overview
    - One-to-All Broadcast & All-to-One Reduction
    - All-to-All Broadcast & All-to-All Reduction
    - **All-Reduce & Prefix-Sum**
    - Scatter & Gather
    - All-to-All Personalized Communication

All-Reduce

Messages M^i exist on each process *i*, and we want to combine these messages in the same way on each processor.



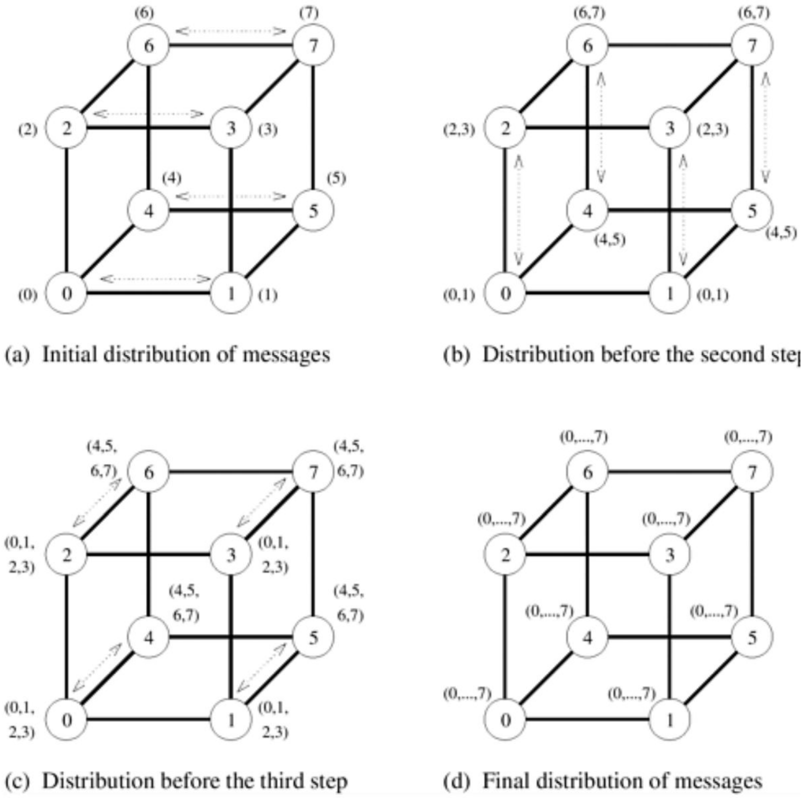
What is the faster way of performing All-Reduce on a Hypercube?

What is the faster way of performing All-Reduce on a Hypercube?

All-to-All Broadcast style communication, except with additional sums after each communication

All-Reduce (Hypercube)

Whiteboard example on how to update All-to-All Broadcast for All-Reduce

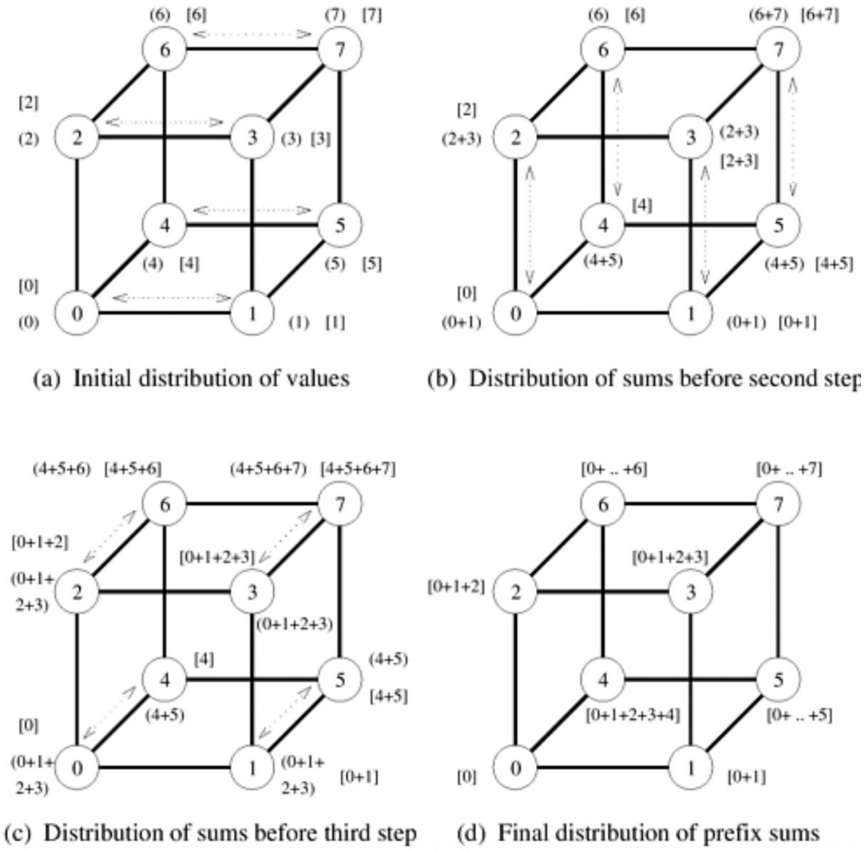


Prefix Sum

An array where each element stores the cumulative sum of all previous elements (useful for histograms, radix sort, etc.

Input array [2, 4, 6, 8] → Prefix sums [2, 6, 12, 20]

Prefix Sum on a Hypercube

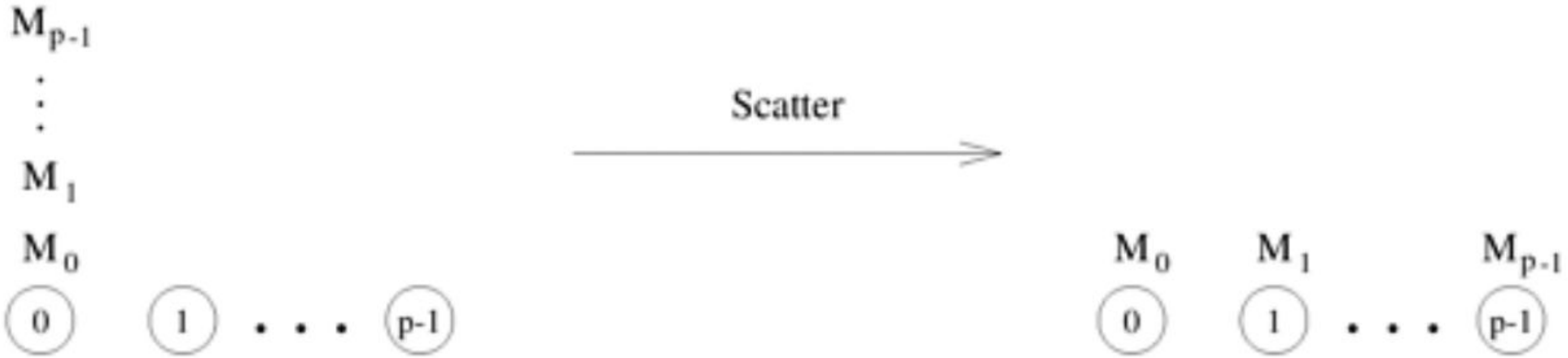


Lecture Overview

- Homework 1
- **Basic Communication Operations**
    - Overview
    - One-to-All Broadcast & All-to-One Reduction
    - All-to-All Broadcast & All-to-All Reduction
    - All-Reduce & Prefix-Sum
    - **Scatter & Gather**
    - All-to-All Personalized Communication

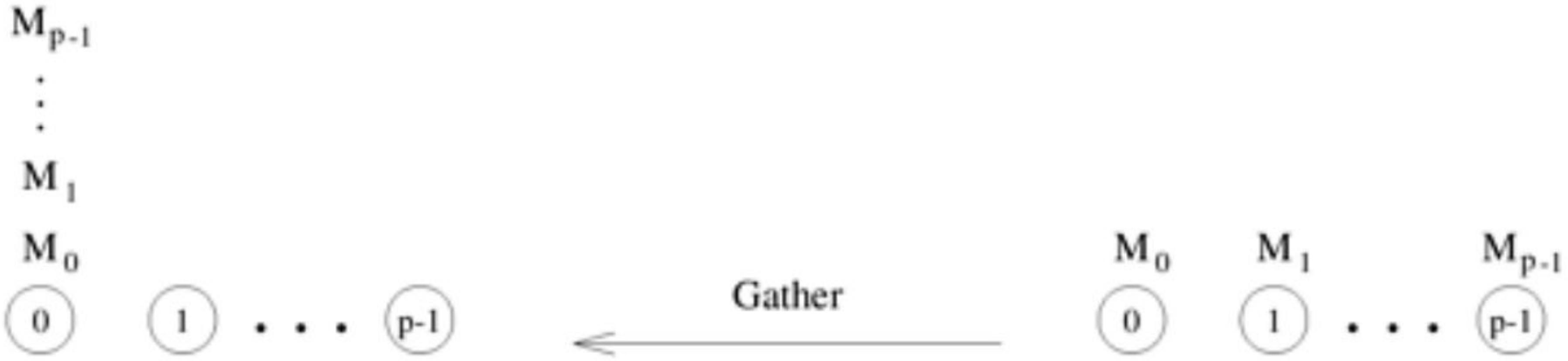
Scatter

A single processor has a separate message *m* for each other processor in the network



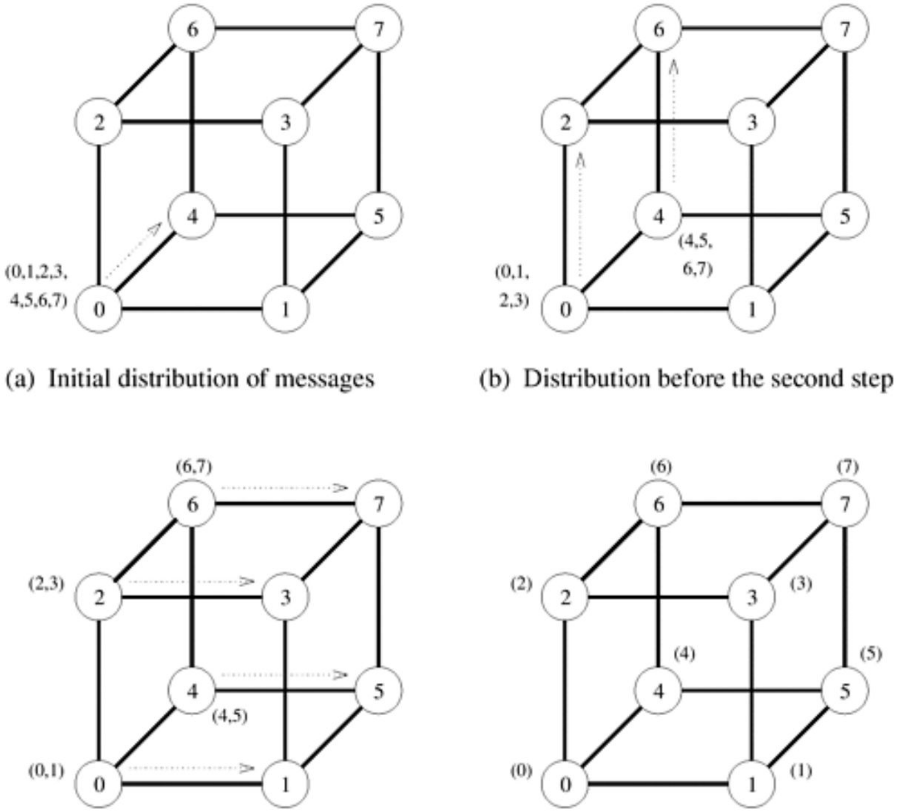
Gather

Each processor has a message *m*, all of which must be collected onto a single processor



Scatter on Hypercube

Quite similar to All-to-One Reduction or One-to-All Broadcast



(a) Initial distribution of messages

(b) Distribution before the second step

(c) Distribution before the third step

(d) Final distribution of messages

Scatter & Gather Cost (Ring/Mesh/Hypercube)

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

Scatter & Gather Cost (Ring/Mesh/Hypercube)

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

How did we arrive at this cost?

Lecture Overview

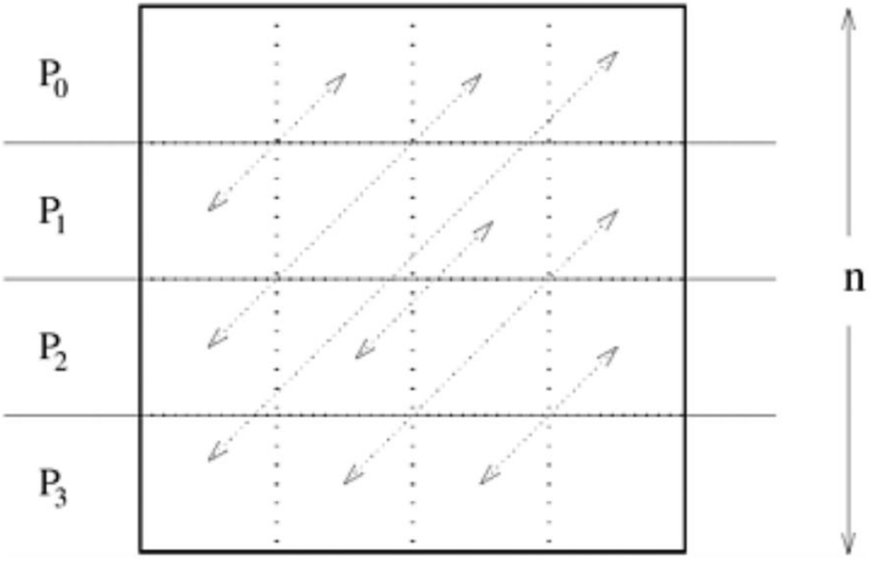
- Homework 1
- **Basic Communication Operations**
    - Overview
    - One-to-All Broadcast & All-to-One Reduction
    - All-to-All Broadcast & All-to-All Reduction
    - All-Reduce & Prefix-Sum
    - Scatter & Gather
    - **All-to-All Personalized Communication**

All-to-All Personalized Communication (AAPC)

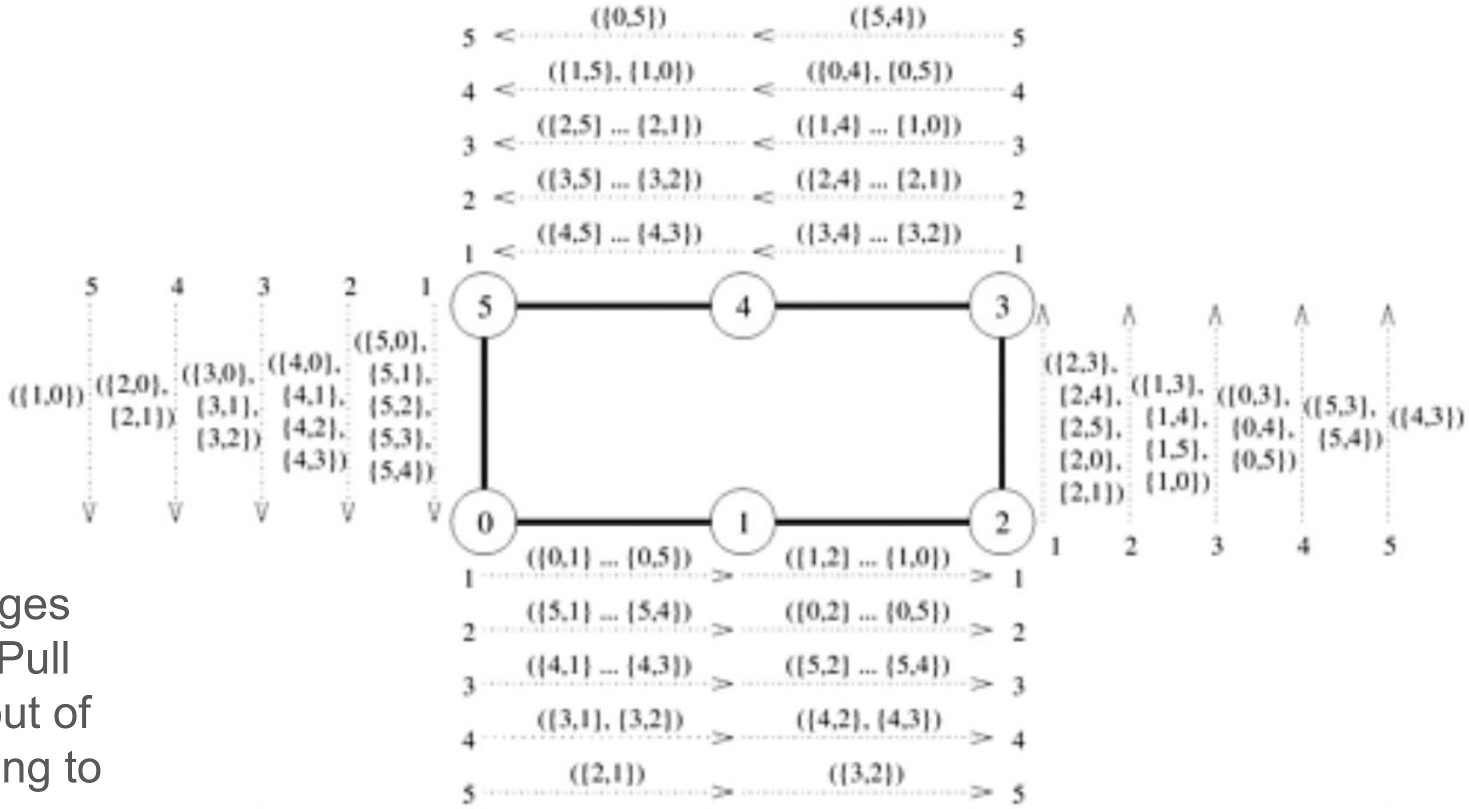
All processors contain a unique message
for each other processor.

AAPC Example (Transposing a Matrix)

Suppose we have some matrix such that each processor holds a separate set of rows (i.e. P_0 has the first set of rows, P_1 the next set of rows, etc.), and we want to transpose the matrix such that P_0 has the first set of columns, P_1 the second set of columns, etc.



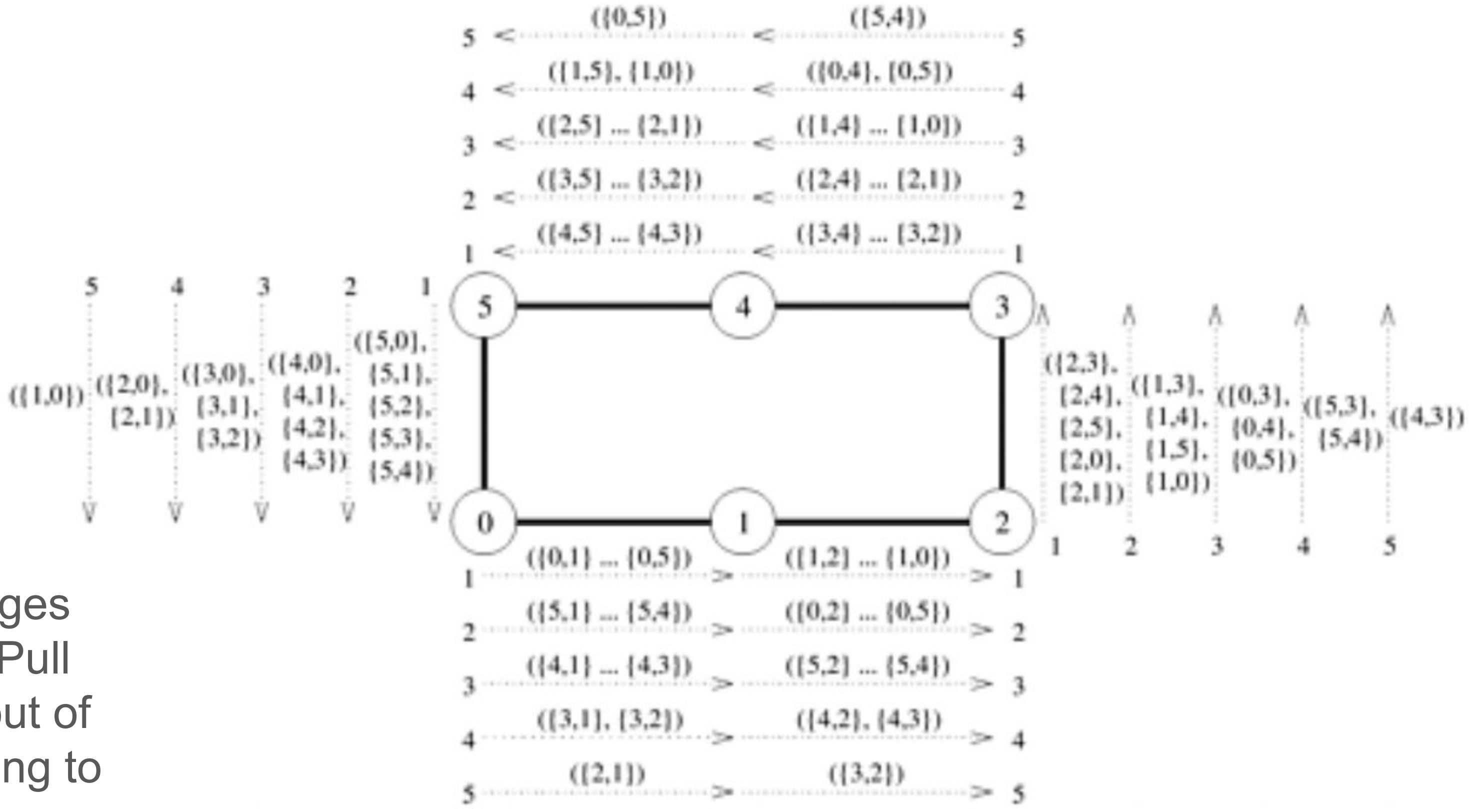
AAPC (Ring)



Pass along messages one link at a time. Pull messages for you out of the set and pass along to next processor

AAPC (Ring)

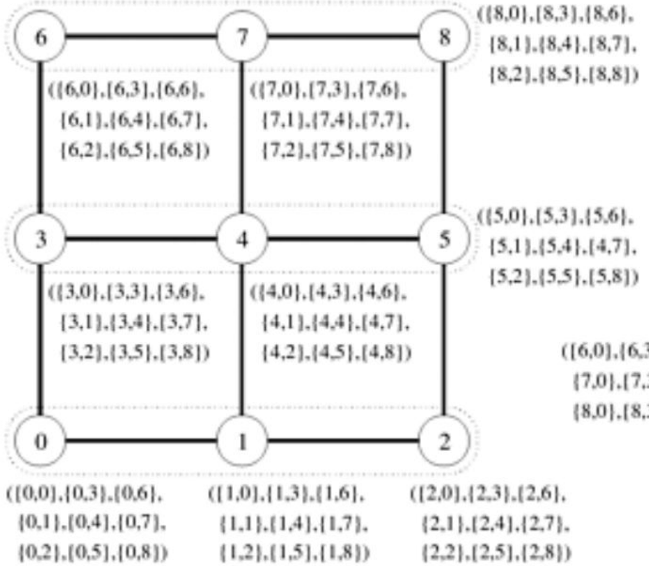
Cost?



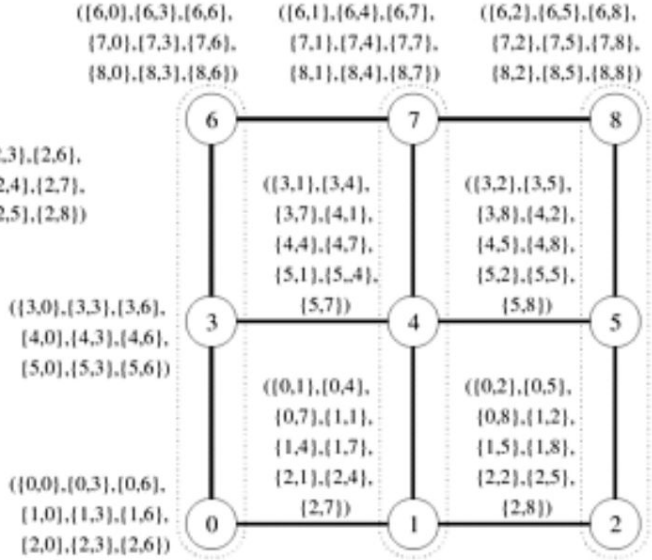
Pass along messages one link at a time. Pull messages for you out of the set and pass along to next processor

AAPC (Mesh)

As before, treat each dimension as mutually exclusive Rings



(a) Data distribution at the beginning of first phase

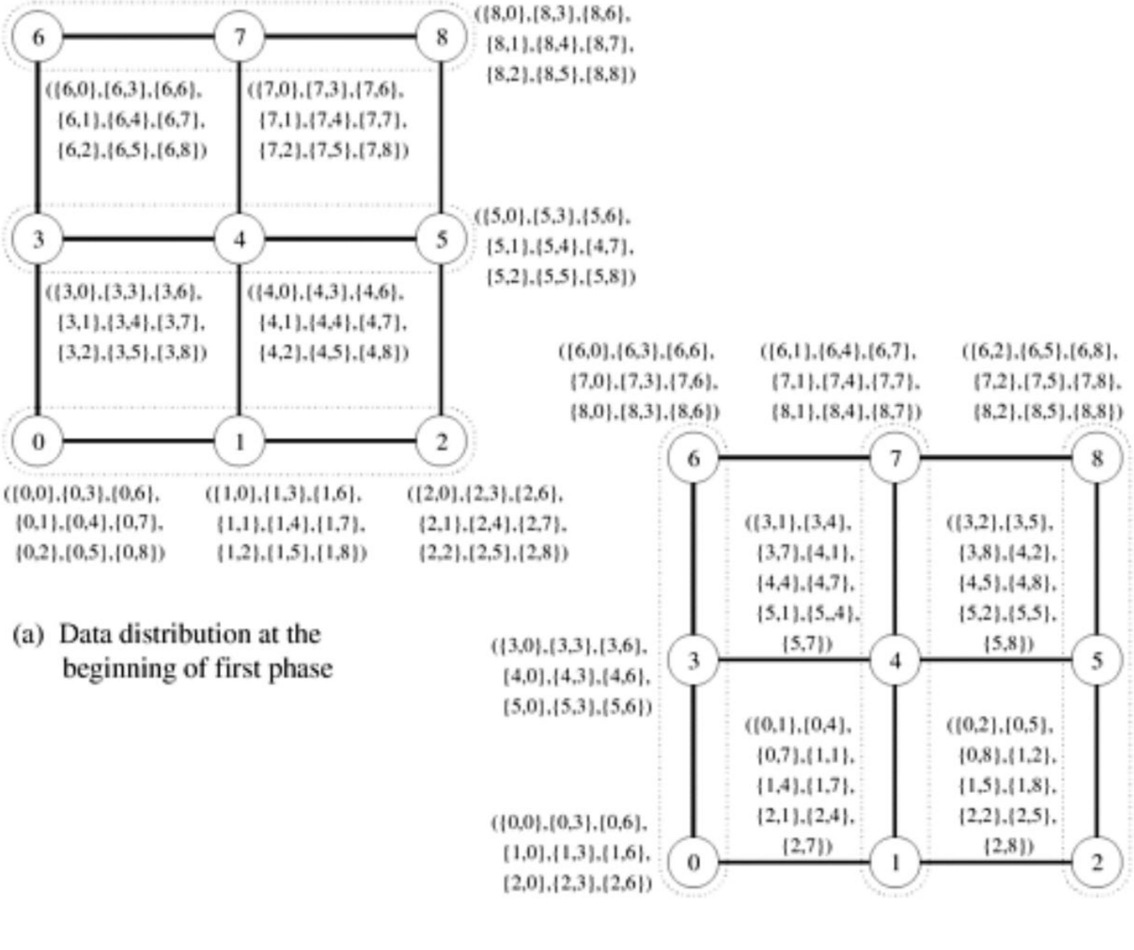


(b) Data distribution at the beginning of second phase

AAPC (Mesh)

Cost?

As before, treat each dimension as mutually exclusive Rings

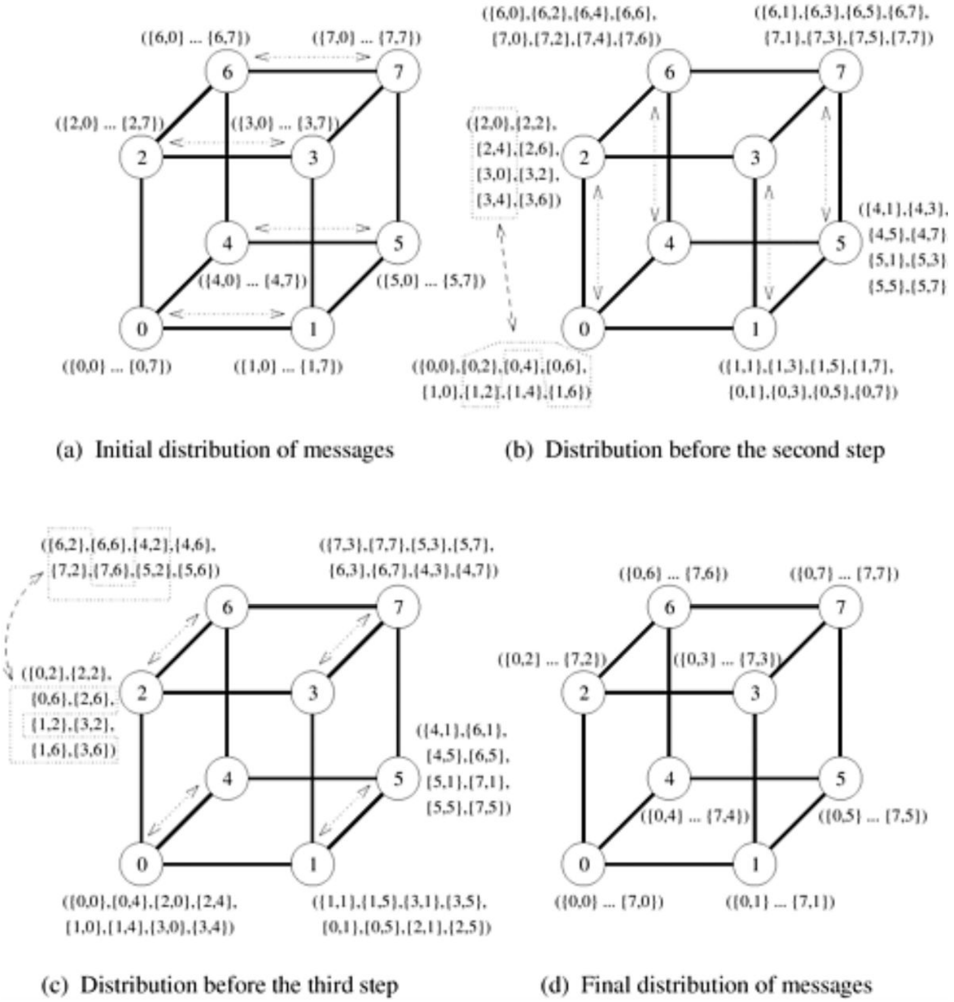


(a) Data distribution at the beginning of first phase

(b) Data distribution at the beginning of second phase

AAPC (Hypercube)

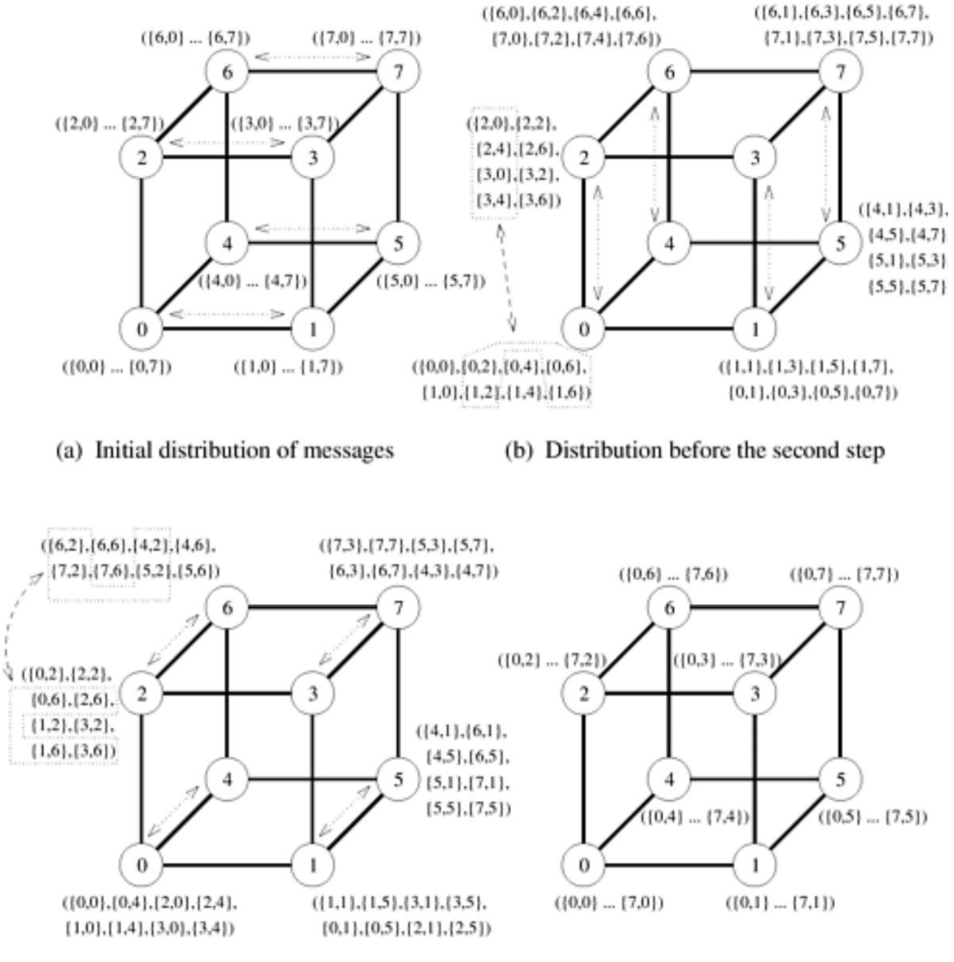
As before, treat each dimension as mutually exclusive 2-processor Rings



AAPC (Hypercube)

Cost?

As before, treat each dimension as mutually exclusive 2-processor Rings



(a) Initial distribution of messages

(b) Distribution before the second step

(c) Distribution before the third step

(d) Final distribution of messages