EE/CSCI 451: Parallel and Distributed Computation

Lecture #12

9/29/2020

Viktor Prasanna

prasanna@usc.edu

ceng.usc.edu/~prasanna

University of Southern California

USC University of Southern California

Outline

Last class

- Analytical modeling
    - Speedup
    - Scalability
    - Efficiency
    - Performance analysis

Today

Communication Primitives (Chapter 4)

- Communication (cost) models
- Definitions of communication primitives
- Example Implementation of some communication primitives (Broadcast)
- Example use of communication primitives

Announcement

- PHW4 out, due 10/9
- HW5 out, due 10/4

Course Project

- Project timeline
  - Week 7-8: Identify team members and project topic, discuss with instructor or TA
  - End of Week 9: Project proposal due (Oct. 16)
  - Week 12-13: Presentation
  - End of Week 13: Project report due (Nov. 13)

- Grading breakdown for the course project
  - Proposal: 25%
  - Final presentation: 25%
  - Final report: 50%

Example Project

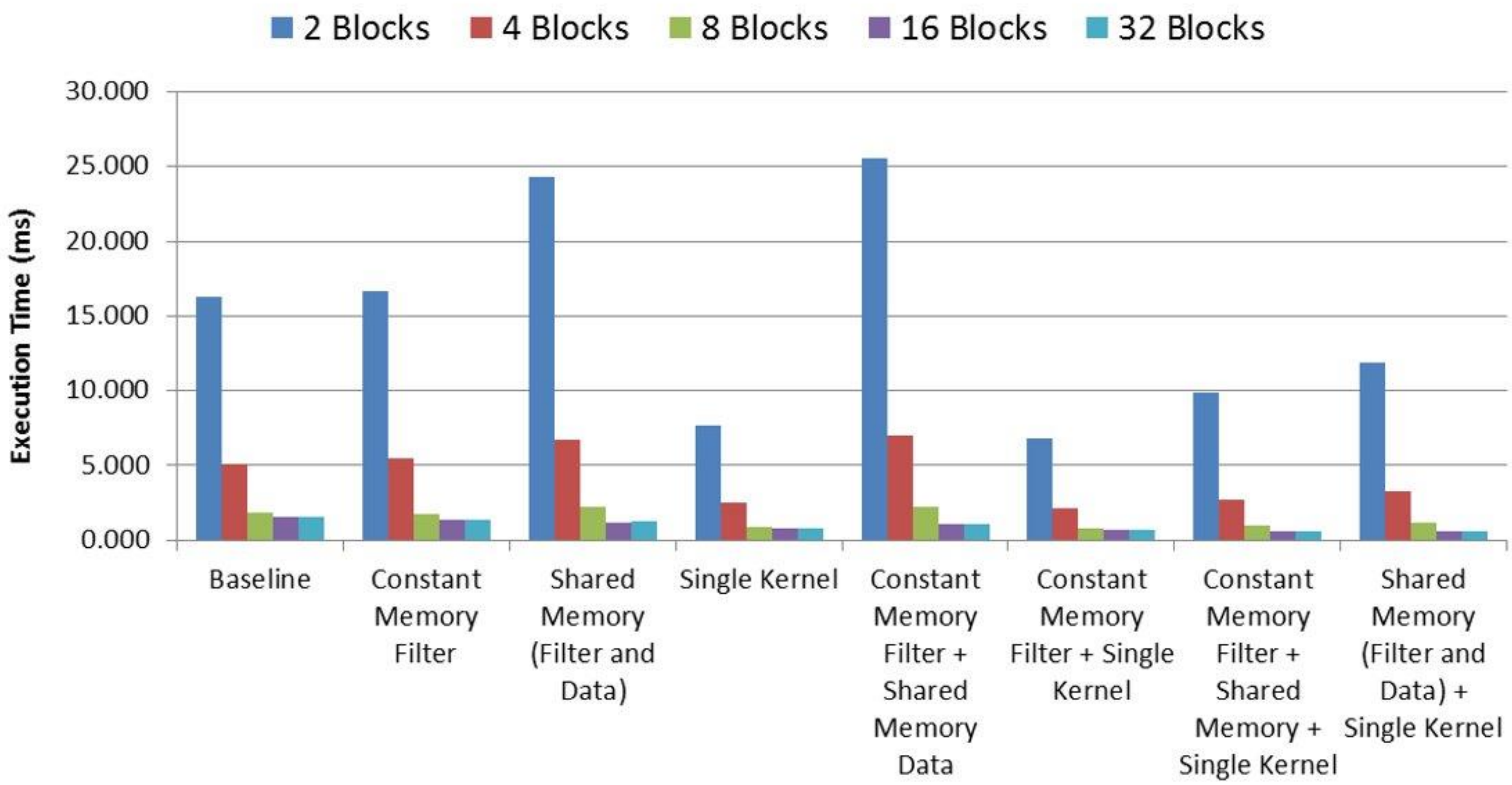
- Design Space Exploration of 2D Image Gaussian Blurring using GPU
- Acknowledgement:
  - Ren Chen, Andrea Sanny and Geoffrey Tran
- Objectives:
  - Implement a parameterized algorithm for Gaussian Blurring of a 2D Image
  - Evaluate the performance of the algorithm with respect to the algorithmic and architectural parameters
- Implement two algorithms
  - Image Separation: Separate the pixels into R, G and B values to create three images
  - Filtering: Image convolution on each of the images
- Algorithm Design Parameters
  - Block Size
  - Grid Size

Project Scope

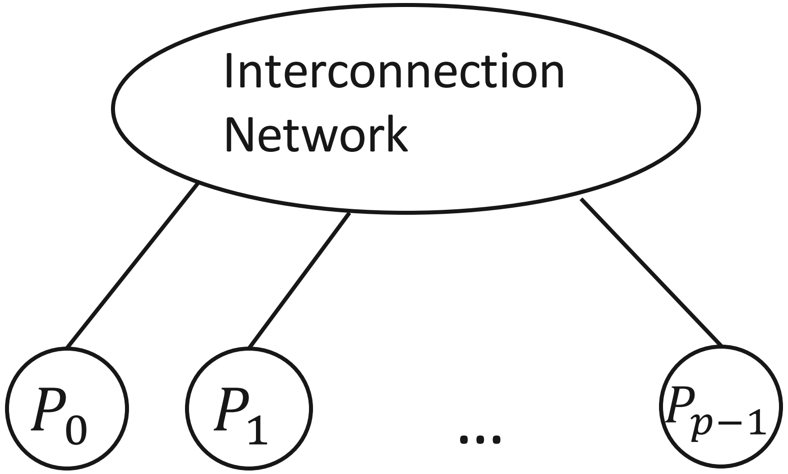
- GPU Architecture Parameters

Baseline	Basic image separation and image processing kernel implemented in CUDA
Constant Memory (Filter)	The filter is stored into constant memory
Shared Memory (Filter and Data)	The image and filter data are both stored into shared memory, managed by the programmer explicitly
Constant Memory (Filter) + Shared (Data)	The filter is stored into constant memory. The image data is stored into shared memory
Single Kernel	The image separation kernel is omitted and each thread processes all three colors per pixel
Constant Memory (Filter) + Single Kernel	The filter is stored into constant memory. The image separation kernel is omitted and each thread processes all three colors per pixel
Shared Memory (Filter and Data) + Single Kernel	The data is stored in shared memory, but the image is processed all at once
Constant Memory (Filter) + Shared Memory + Single Kernel	All three optimizations are utilized

Experimental Results



Basic Communication Operations



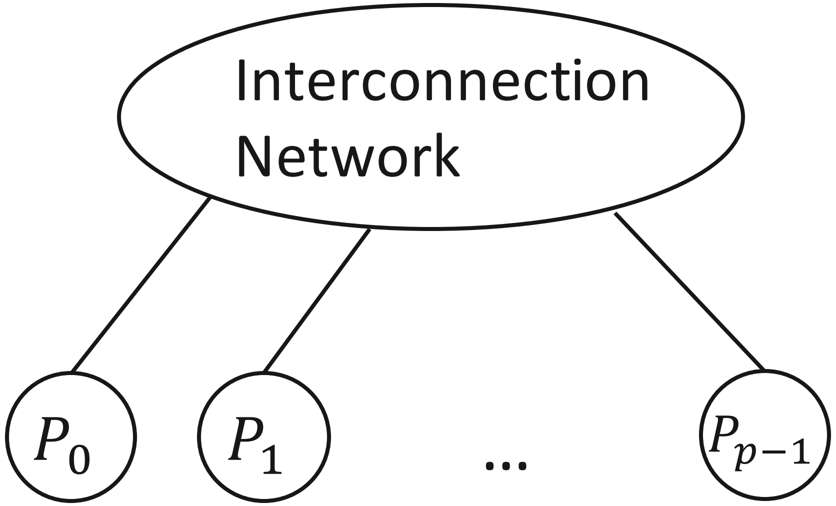
General structure of parallel program

Interprocess communication

Compute using local data
Communicate
Compute using local data
Communicate
.....

Analyzing Communication Cost (1)

1. Simple model



Communication time $= \boldsymbol{t_s + t_w \cdot m}$

t_s = overhead

t_w = per word transfer rate

m = message size

Note:

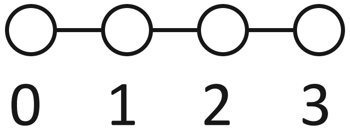
>1 Source destination pair may communicate at the same time, can result in congestion, affects t_w

Analyzing Communication Cost (2)

2. Network model

Ex. 1-D Mesh

$$i \rightarrow i+1,\ \ i-1$$



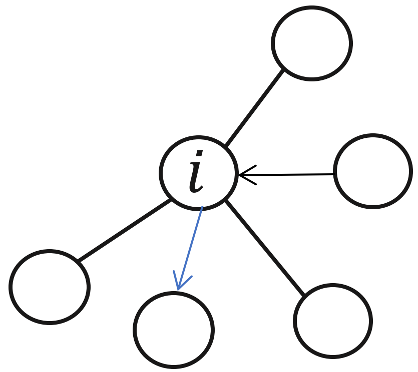
Synchronous model

Unit of time:

- Local operation
- Unit data communication using a link

Analyzing Communication Cost (3)

Single port communication



Nodes can have > 1 link

During a communication step, each node

- can send a message using at most one link
- can receive a message using at most one link

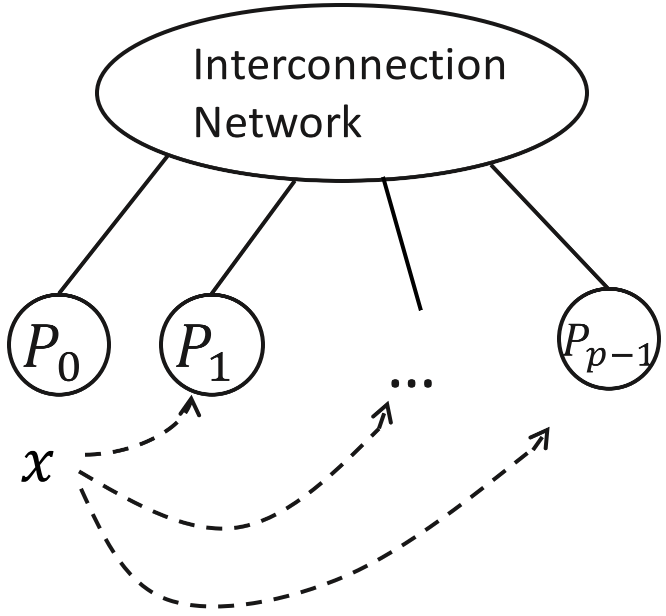
A node can send and receive during a communication step (may be using the same link)

Note: Multi port communication model can be similarly defined

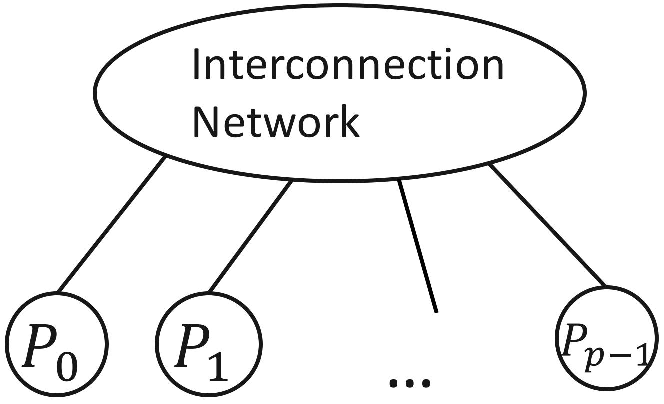
Some Communication Primitives (1)

Broadcast and Reduction

One-to-all broadcast



All-to-one reduction



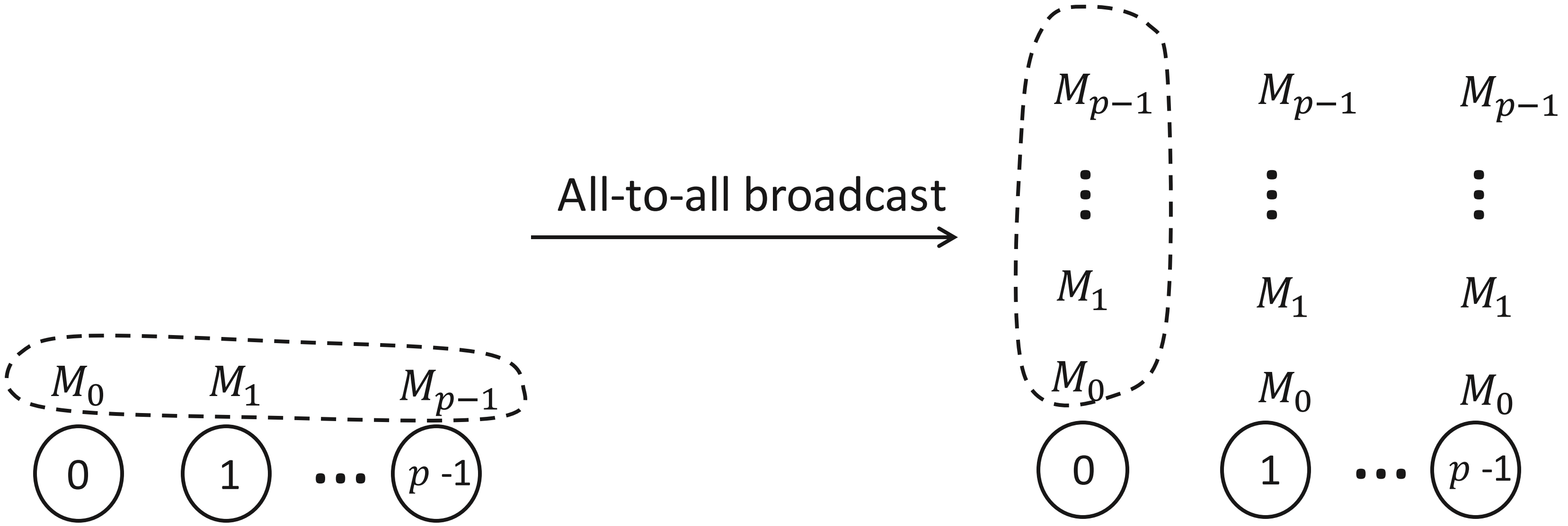
Output$= \sum_{i=0}^{p-1} x_i$

Reduction operation = sum, min, …associative op.

$(a + b) + c = a + (b + c)$

Some Communication Primitives (2)

All-to-all broadcast



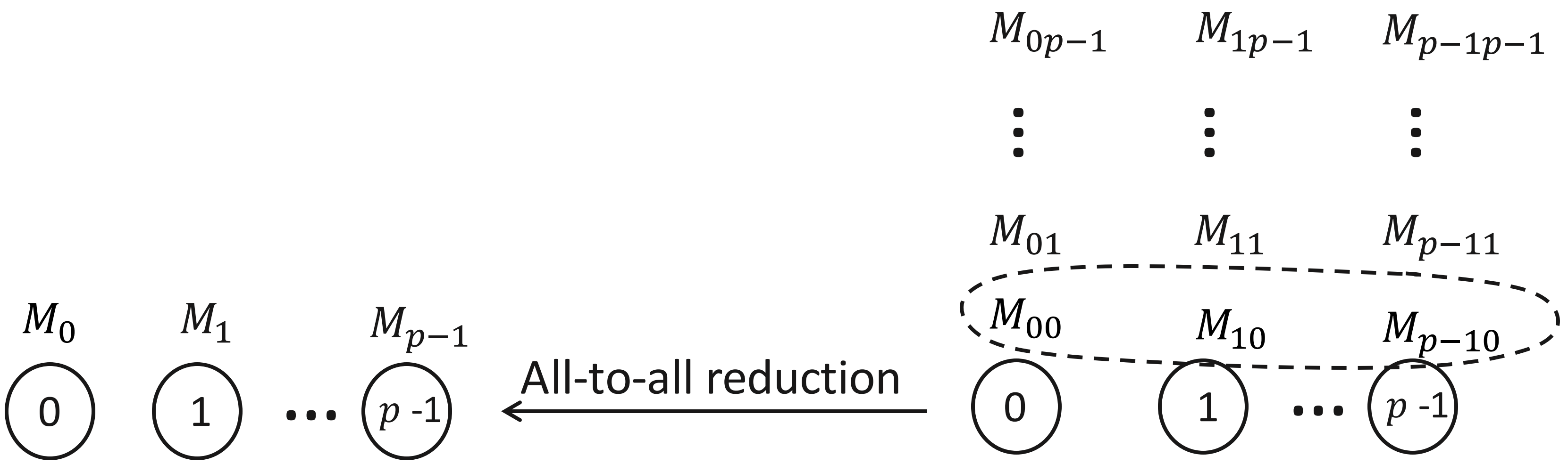
Total # of input messages $= p$

Each process receives each of the p messages

Total output size $= p^2$

Some Communication Primitives (3)

All-to-all reduction



$$M_i = \sum_{j=0}^{p-1} M_{ji}$$

Total # of inputs $= p^2$

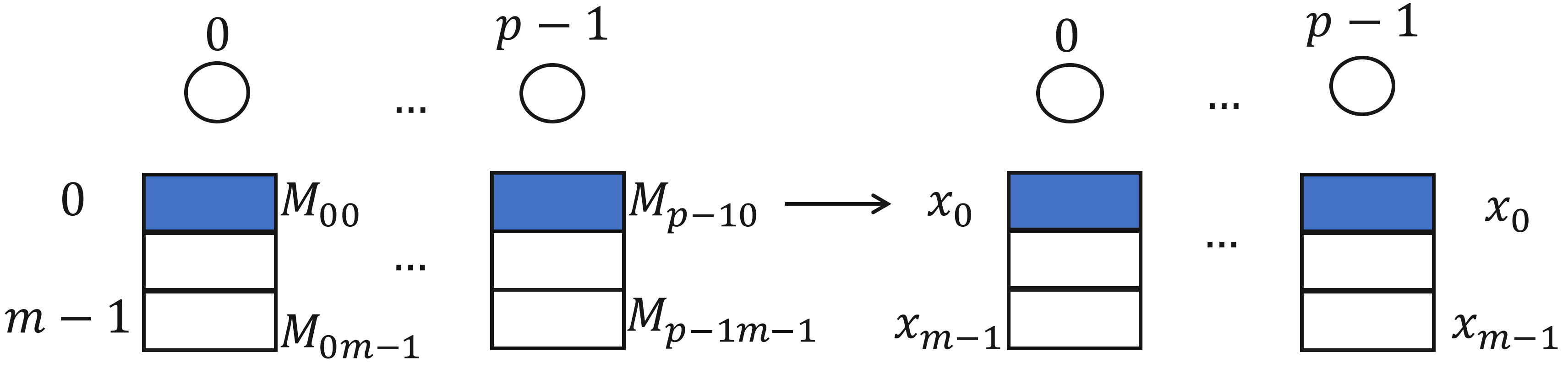
Total # of outputs $= p$

Some Communication Primitives (4)

All-Reduce

Reduction Operation: message of size m (vector of size m)

copy result (of size m) to all processes



$$x_i = \sum_{j=0}^{p-1} M_{ji}$$

$0 \le i < m$

Note:

- Total # of distinct outputs $= m$
- Each process has a copy at the end

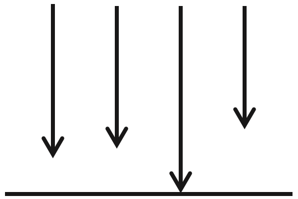
Some Communication Primitives (5)

Use of All-Reduce

$m = 1$ (one message per process)

Note: All-Reduce can be implemented as

- All-to-one Reduce
- One-to-all Broadcast



BARRIER SYNC among p processes

Each process — After local computation

All-Reduce (data)

All-Reduce completes only after each process contributes a data value

Some Communication Primitives (6)

Prefix Sum (Scan)

Input: $x_0 \quad x_1 \quad \ldots \quad x_{p-1} \quad (P_i \text{ has } x_i)$

Output: $y_0 \quad y_1 \quad \ldots \quad y_{p-1} \quad (P_i \text{ has } y_i)$

$$y_i = \sum_{j=0}^{i} x_j \qquad 0 \leq i < p$$

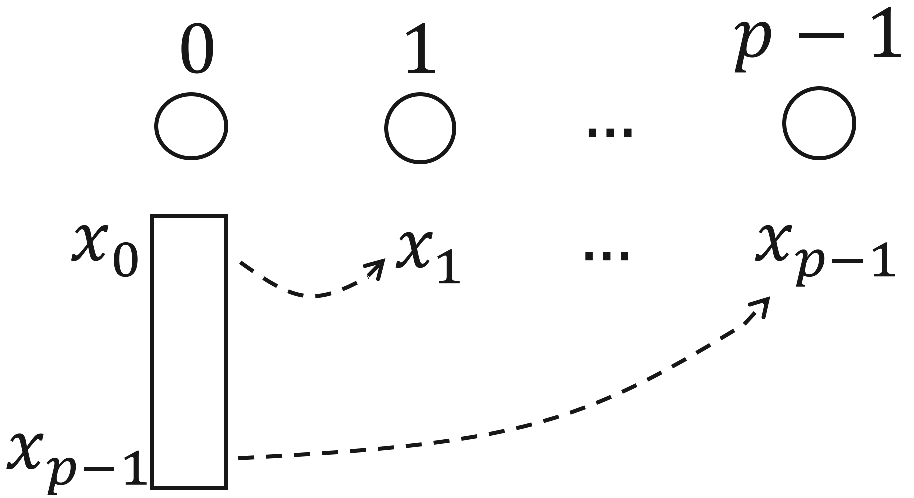
Ex:								
	0	1	2	3	4	5	6	7
	0	1	3	6	10	15	21	28

Some Communication Primitives (7)

Scatter and Gather

Scatter: one process sends a **distinct** message to every other process

One-to-all personalized communication

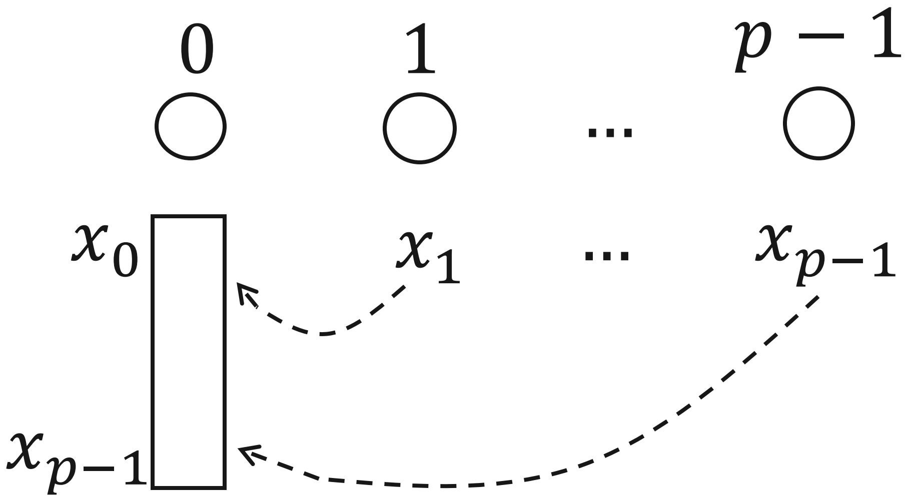


Some Communication Primitives (8)

Scatter and Gather

Gather: a single process collects a distinct message from every other process

All-to-one personalized communication



Some Communication Primitives (9)

All-to-all personalized communication

- Each process sends a distinct message to all the processes (processors or nodes)

- Note: a message can be empty

- Total # of input messages $= p^2$

- $p =$ Total number of processes

Some Communication Primitives (10)

Example

$$
p\left\{\overbrace{\boxed{\begin{array}{cccc}
M_{0,0} & M_{1,0} & \vdots & M_{p-1,0} \\
M_{0,1} & M_{1,1} & & M_{p-1,1} \\
 & \cdots & & \cdots \\
M_{0,p-1} & M_{1,p-1} & & \cdots
\end{array}}}^{p}\right.
$$

$M_{i,j}$ = message to be sent from P_i to P_j $(0 \leq i, j < p)$

Some Communication Primitives (11)

Example (cont.)

$M_{i,j}$ = message to be sent from P_i to P_j $(0 \leq i, j < p)$

Some Communication Primitives (12)

Permutation

Each process has a data

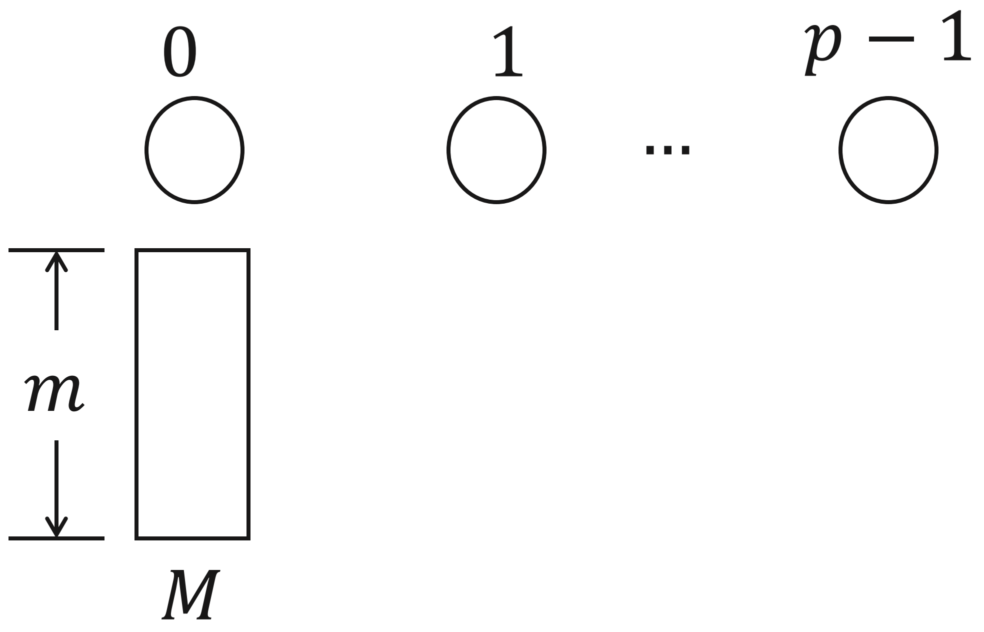
Sends data to one destination process

Set of destination processes $=$ permutation of $0, 1, \ldots, p-1$

Example: Circular shift right

Process i sends to $(i+1) \bmod p$

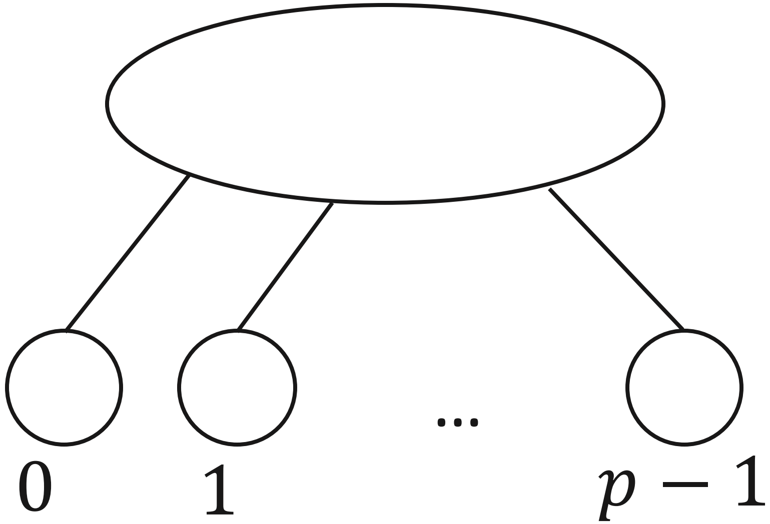
Performing One-to-all Broadcast



Broadcast to all processes

At the end: each process has a copy of M

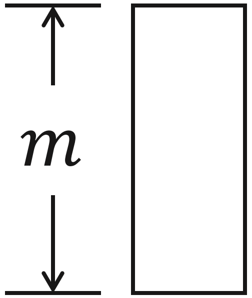
One-to-all Broadcast (1)



Do $i = 1$ to $p - 1$

Send from P_0 to P_i

End



Serial !

Total (communication) time $= (p - 1)(t_s + t_w \cdot m)$

One-to-all Broadcast (2)

Recursive doubling

In i^{th} iteration ($0 \leq i < \log_2 p$)

- processors that differ in index by 2^j communicate, $j = \log_2 p - 1 - i$

- 2^i processors (that have not received data before) receive data

One-to-all Broadcast (3)

Example: $(p = 8)$

$(i = 0)$ $j = 2$ $0 \rightarrow (0 + 2^2) = 4$

$(i = 1)$ $j = 1$ $0 \rightarrow (0 + 2^1) = 2$

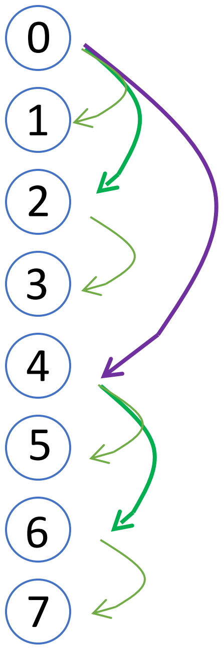
$4 \rightarrow (4 + 2^1) = 6$

$(i = 2)$ $j = 0$ $0 \rightarrow (0 + 2^0) = 1$

$2 \rightarrow (2 + 2^0) = 3$

$4 \rightarrow (4 + 2^0) = 5$

$6 \rightarrow (6 + 2^0) = 7$



One-to-all Broadcast (4)

Do $i = 0$ to $\log_2 p - 1$

$j = \log_2 p - 1 - i$

In parallel do:

If $idx = k \cdot 2^{j+1}$ for some $k \in N$

P_{idx} send data to P_{idx+2^j}

End parallel

BARRIER

End

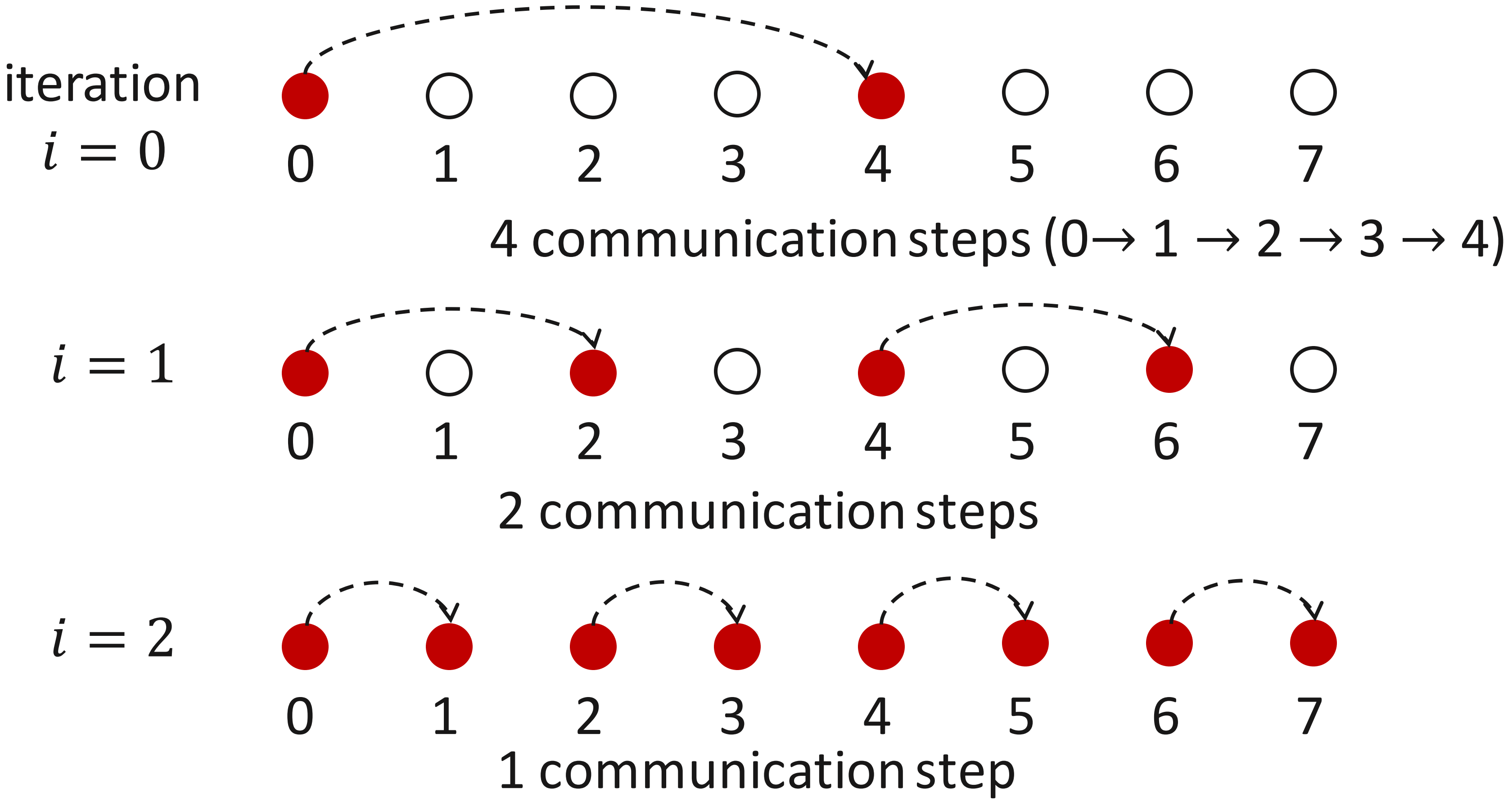
N = set of natural numbers = $\{0,1,\ldots\}$

Total (communication) time = $(\log p)\,(t_s + t_w \cdot m)$

m = message size

One-to-all Broadcast (5)

On a linear array (network model)



One-to-all Broadcast (6)

On a Linear Array (Network Model)

$\frac{p}{2}$ steps per message of size 1 (Simple approach)

Total communication time $= \left(\frac{p}{2} + \frac{p}{4} + \cdots + 1\right) \cdot m$

$= O(p \cdot m)$

Communication time when using p processors

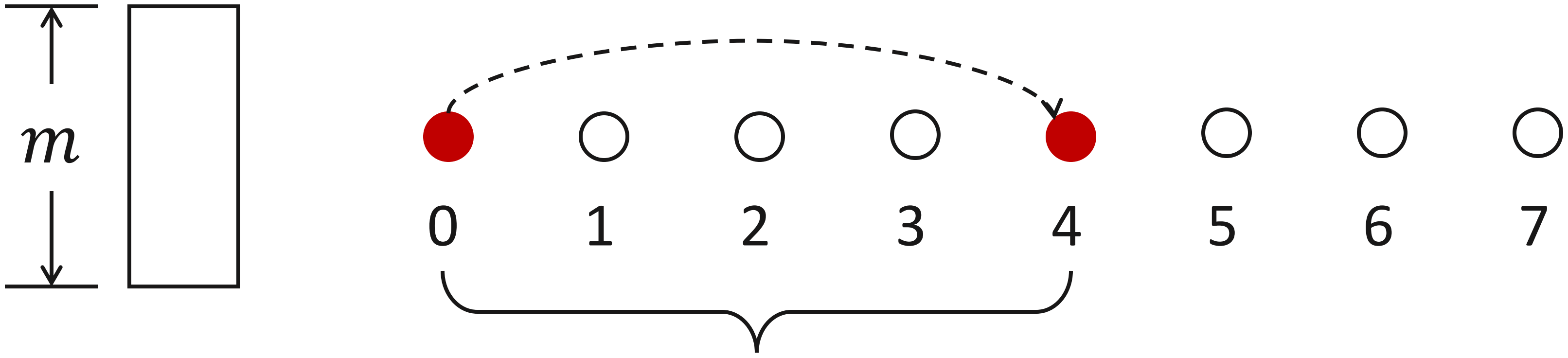
$$T_p(m) = T_{p/2}(m) + \left(\frac{p}{2}\right) \cdot m$$

$$T_2(m) = m$$

One-to-all Broadcast (7)

On a Linear Array

Can pipeline sending message of size m



Pipeline

Time $= \left({}^{p}/_{2} + m - 1\right)$ to send message of size m from P_0 to $P_{p/2}$

One-to-all Broadcast (8)

Pipelined implementation on a Linear Array (Network model)

$$T_p(m) = T_{p/2}(m) + (\frac{p}{2} + m - 1)$$

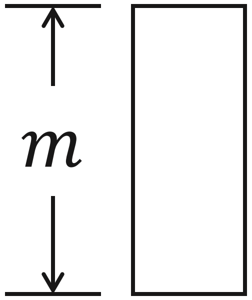
$$T_{p/2}(m) = T_{p/4}(m) + (\frac{p}{4} + m - 1)$$

$$\vdots$$

$$\text{Total time} = O(p + m \cdot \log_2 p)$$

One-to-all Broadcast (9)

Implementation on Linear Array (Simple pipelined implementation)

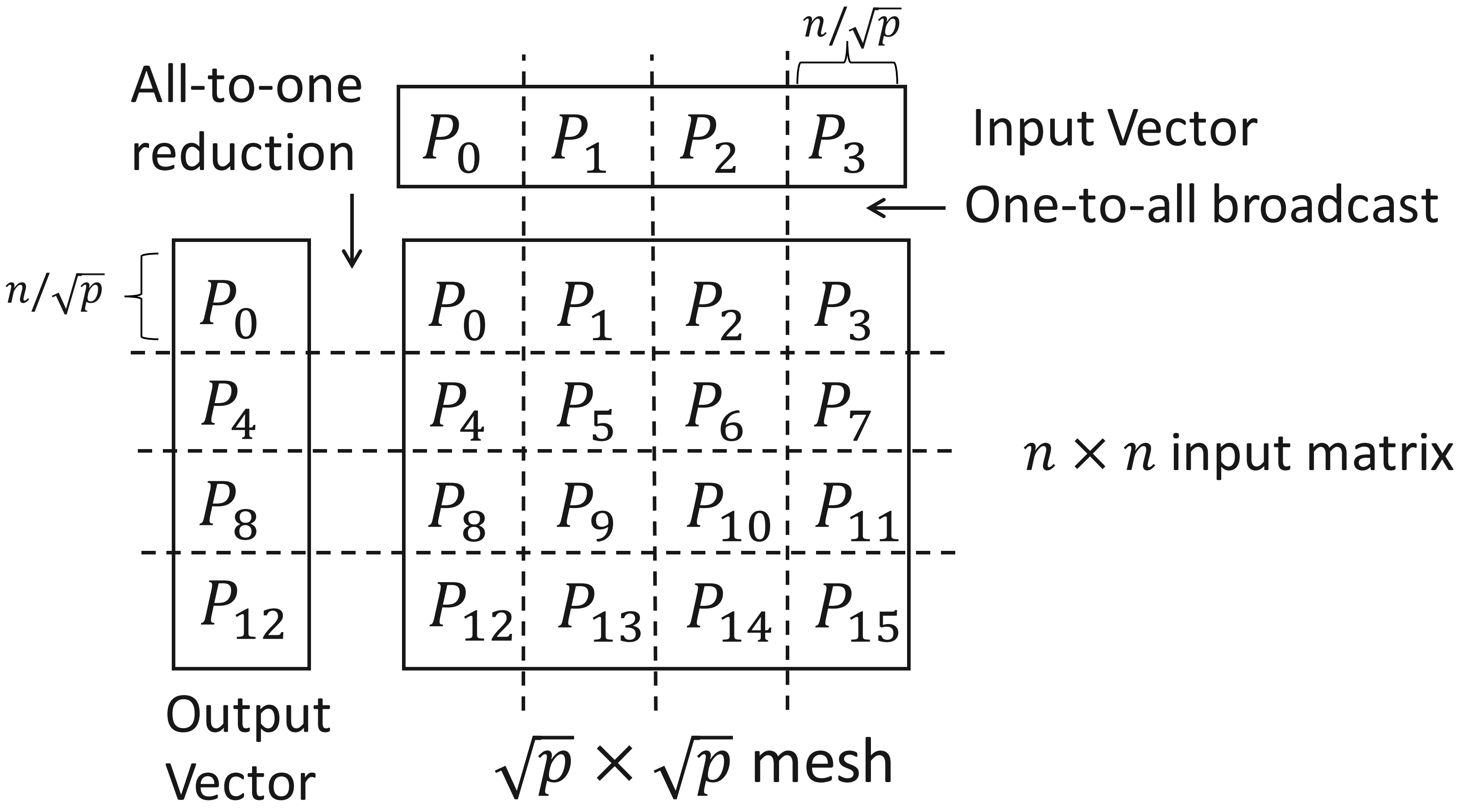


- Receive unit data from left
- Make local copy
- Send to right

$(p-1)+(m-1)$ steps

Example Parallel Algorithm (1)

Matrix Vector Multiplication



Example (2)

Processor (i, j) has $\frac{n}{\sqrt{p}} \times \frac{n}{\sqrt{p}}$ input matrix

One-to-all broadcast along each column of data of size $\frac{n}{\sqrt{p}}$

Perform matrix vector product in each PE ($^{n}/_{\sqrt{p}}$ outputs)

All-to-one Reduction in each row (message size = $^{n}/_{\sqrt{p}}$)

Example (3)

Analysis using Network model

$$\text{Total time} = \left(\frac{n}{\sqrt{p}} + \sqrt{p} - 1\right) \quad \text{Broadcast}$$

$$+\left(\frac{n}{\sqrt{p}}\right)^2 \quad \text{Compute}$$

$$+\left(\frac{n}{\sqrt{p}} + \sqrt{p} - 1\right) \quad \text{Reduction}$$

If $p = n^2$

$$\text{Total time} = O(n)$$

Application Specific Architectures

Can eliminate broadcast: Systolic Arrays

Implementation using VLSI (ASIC), FPGA

Local connections (eg. 2-D Mesh)

- Compute (using local data)
- Communicate (locally) – No broadcast

Signal, Image Processing

$n \times n$ array, $O(n^3)$ serial complexity

$\longrightarrow$ $O(n)$ parallel complexity

Summary

- Communication primitives as building blocks

- (Message Passing) Parallel Algorithm = Compute
  Communicate
  Barrier
  Compute
  Communicate
  …

- Communication Primitives
  1. One-to-all broadcast, All-to-one reduction
  2. All-to-all broadcast, All-to-all reduction
  3. All-Reduce
  4. Prefix Sum
  5. Scatter and Gather
  6. All-to-all personalized communication
  7. Permutation