EE/CSCI 451: Parallel and Distributed Computation

Lecture #13

10/1/2020

Viktor Prasanna

prasanna@usc.edu

ceng.usc.edu/~prasanna

University of Southern California

Announcement

- PHW4 due 10/9
- HW5 due 10/4
- HW3 & 4 grades are out

HW3 Statistics	
Average	90
Median	94.0
Standard Deviation	14

HW4 Statistics	
Average	91.4
Median	96.0
Standard Deviation	8.8

- Midterm 1 solution is posted on Piazza
    - Grades and statistics will be out by next Tuesday

Outline

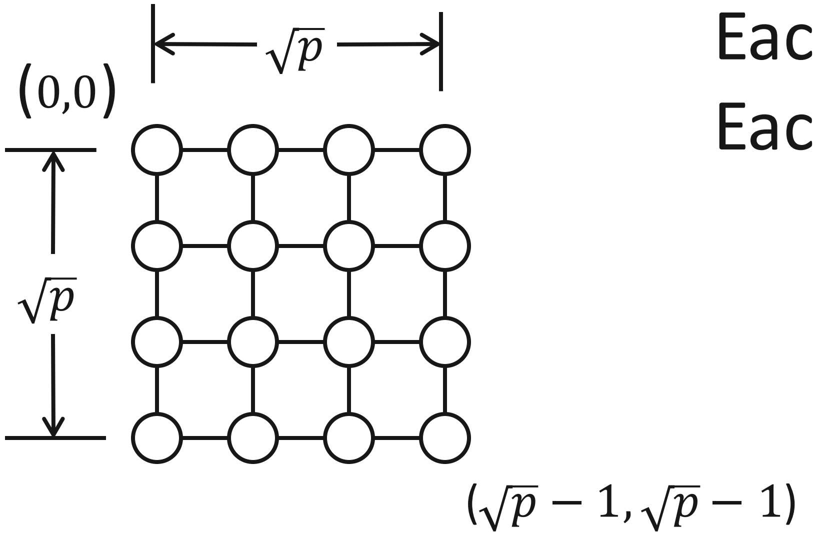
Last class

- Communication primitives (Chapter 4)
  - Communication (cost) models
  - Definitions of communication primitives
  - Example implementation of some communication primitives

Today

- Communication primitives implementation analysis
  - One-to-all broadcast
  - Scatter and gather
- Example parallel programs using primitives
  - Matrix vector multiplication
  - Matrix multiplication
  - Sorting on a linear array
  - K-means clustering

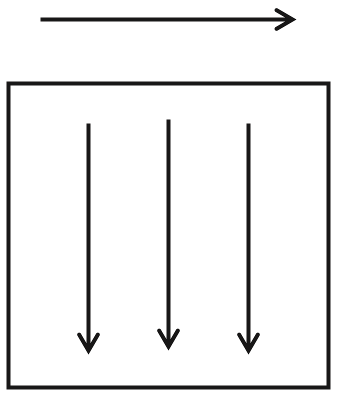
One-to-all Broadcast in a 2-D Mesh (1)



Each row = 1-D Mesh
Each column = 1-D Mesh

$P_{00} \rightarrow$ all processes

One-to-all Broadcast in a 2-D Mesh (2)

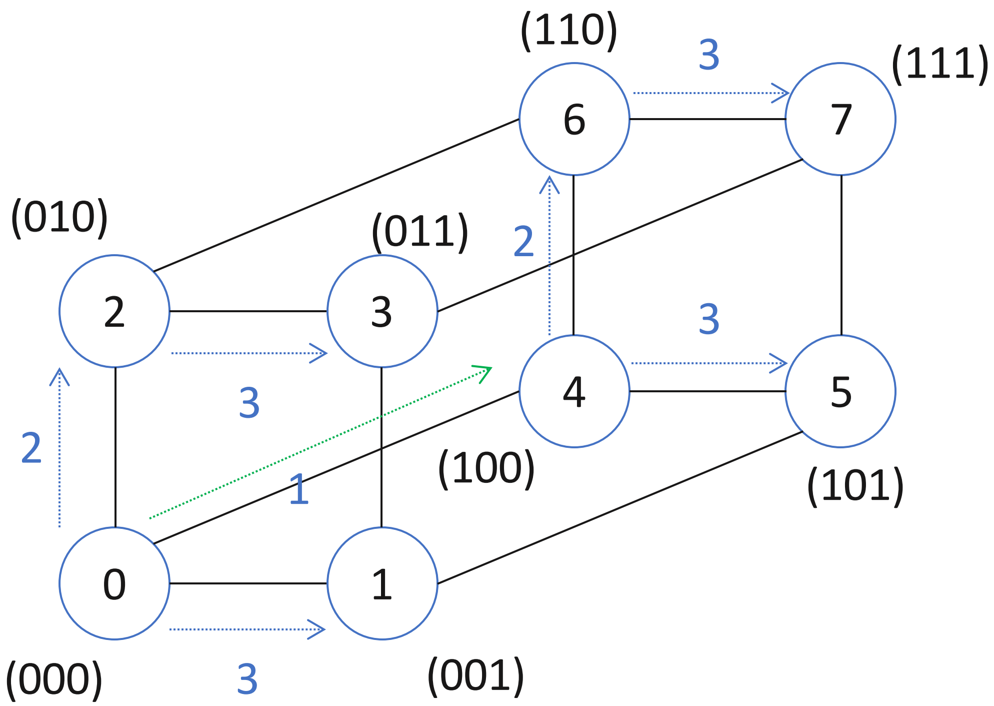


- Broadcast in Row_0
- P_{0j} broadcasts along column j in parallel for all j, $0 \leq j < \sqrt{p}$

($\sqrt{p}$ concurrent broadcasts)

Total time $= (\sqrt{p} - 1 + \sqrt{p} - 1)$ in the Network Model

One-to-all Broadcast in a Hypercube (1)



P_0 to all processes

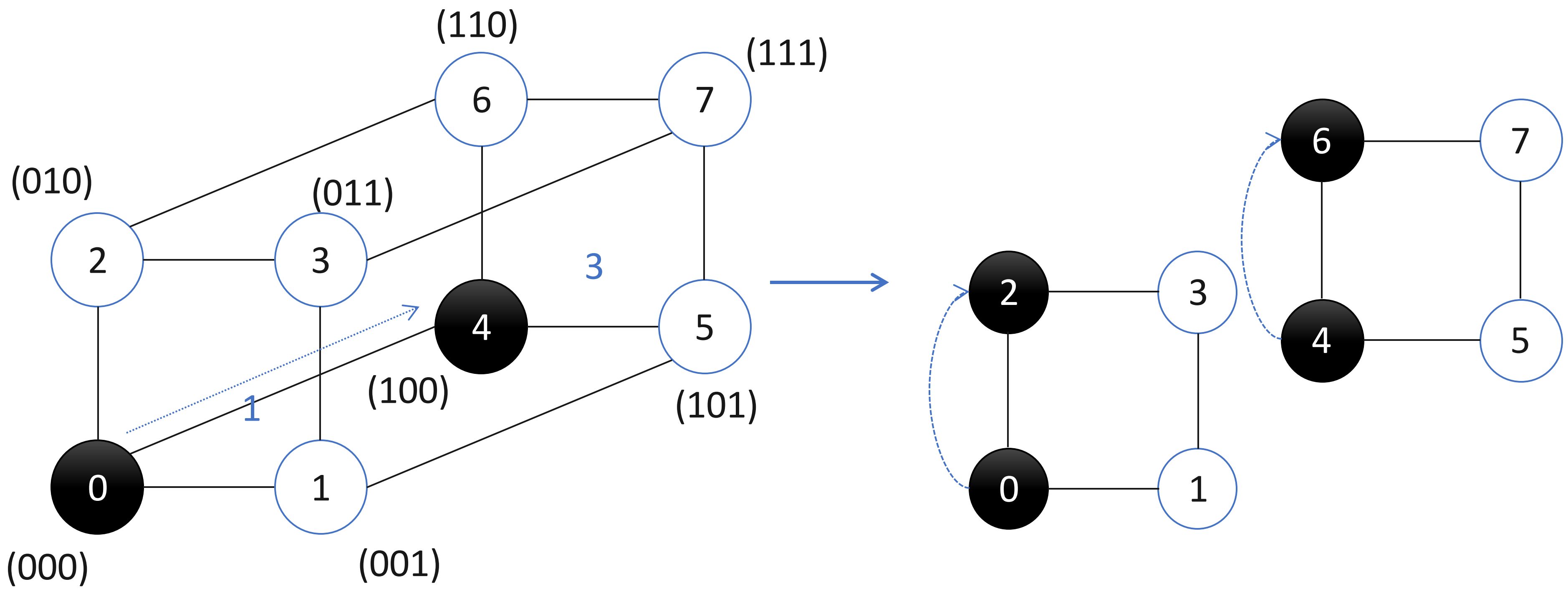
Recursive doubling

Copy from P_0 to $P_{n/2}$

Two hypercube of size $n/2$

Recursively broadcast

One-to-all Broadcast in a Hypercube (2)



One-to-all Broadcast in a Hypercube (3)

Network Model, message size = $\boldsymbol{m}$

Time taken by using p processes

Message size = m

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

$$T_2(m) = m$$

$$T_p(m) = O(m \log p)$$

One-to-all Broadcast in a Hypercube (4)

Prefix/Suffix

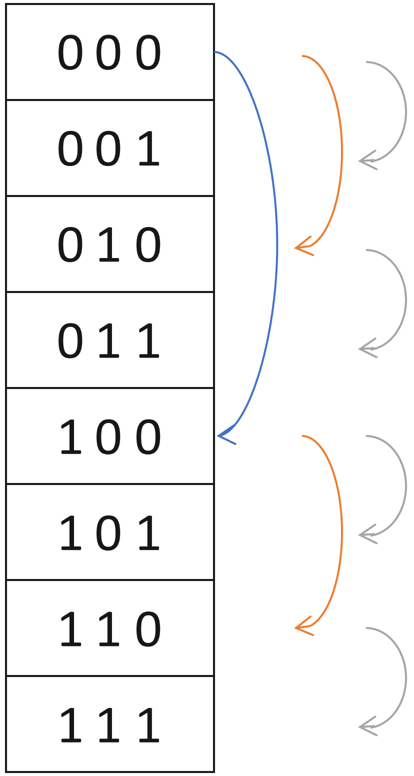
$p = 8$

	2	1	0	← bit index
$i =$	0	1	0	

$$i = \underbrace{0}_{\substack{\textbf{1 bit prefix} \\ \textbf{0}}}\ \underbrace{1\ \ 0}_{\substack{\textbf{2 bits suffix} \\ \textbf{10}}}$$

One-to-all Broadcast in a Hypercube (5)

Implementation of recursive doubling



Do $j = \log p$ to 1

i' = j bit suffix of Processor ID i

if i' = $\underbrace{0 \dots 0}$ then

j bits

P_i sends to $P_{i+2^{j-1}}$ in parallel

End

Example: $p = 8$

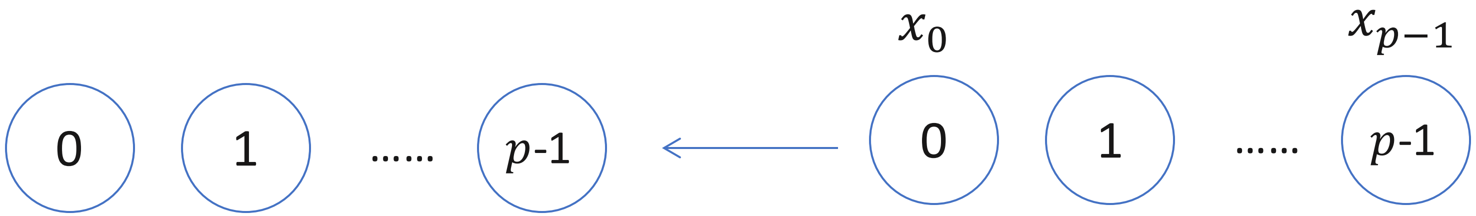
$\longleftarrow j = 3 \longrightarrow$

0	0	0

$\leftarrow j = 2 \rightarrow$

In iteration j, $2^{\log_2 p - j}$ processes active (send message)

All-to-one Reduction



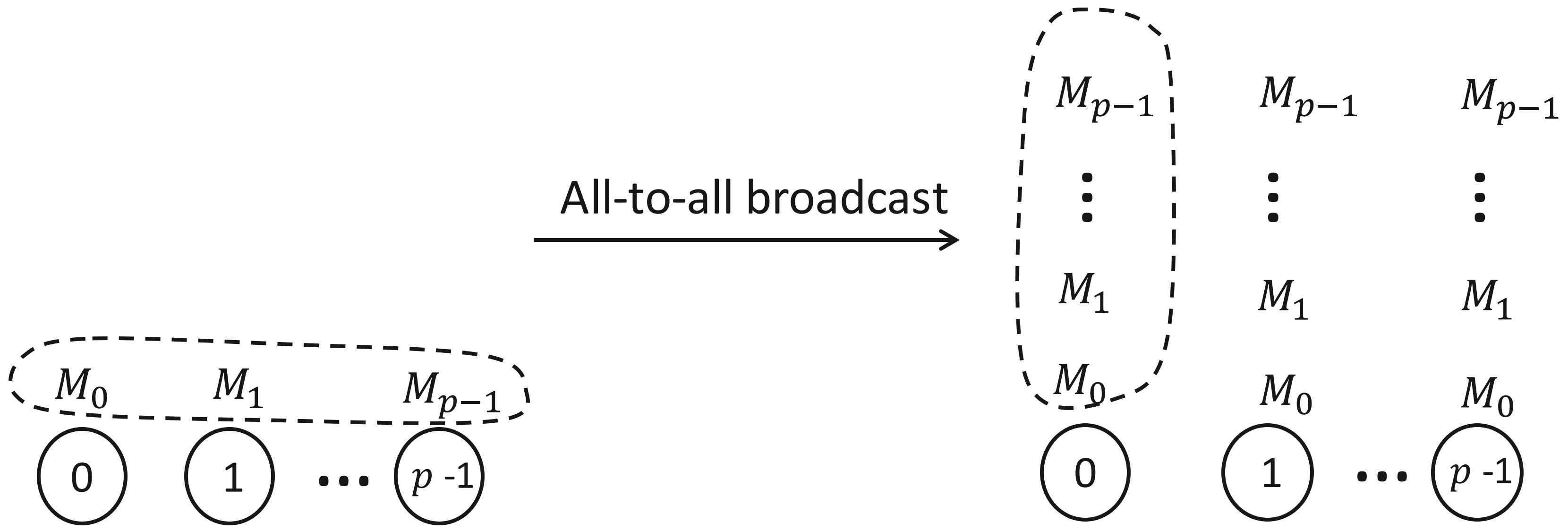
At the end, P_0 has $\sum_{i=0}^{p-1} x_i$

Reduction operator

Dual of One-to-all Broadcast

Hypercube with p nodes $=$ $O(\log p)$ time

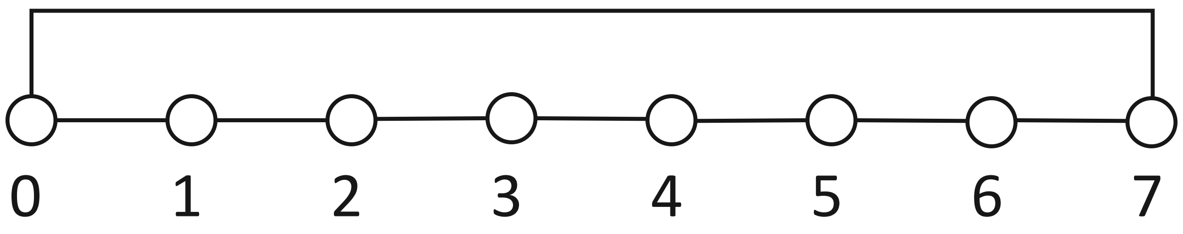
Performing All-to-all Broadcast (1)



Message size = 1

Performing All-to-all Broadcast (2)

On a Linear Array with wrap around (Ring) (Network model)



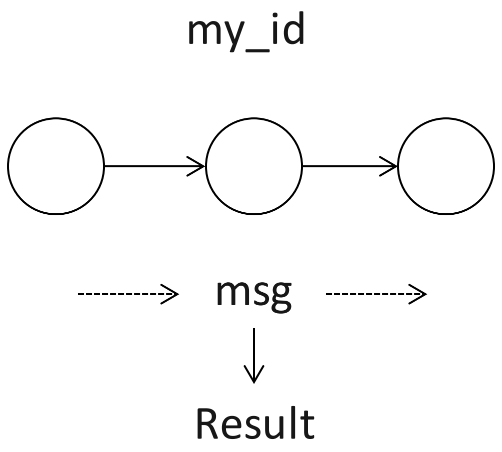
Idea: Cyclic Shift (Right)

Make a local copy

Performing All-to-all Broadcast (3)

Algorithm on a p-node ring

```
procedure ALL_TO_ALL_BC_RING(my_id, my_msg, p, result)
begin
    left := (my_id – 1) mod p;
    right := (my_id + 1) mod p;
    result := my_msg;
    msg := result;
    for i := 1 to p – 1 do
        send msg to right;
        receive msg from left;
        result := result ∪ msg;
    endfor;
end ALL_TO_ALL_BC_RING
```



Performing All-to-all Broadcast (4)

Total parallel time $= O(p)$

(Time) Optimal on a linear array?

(Time) Optimal on any network?
(Single port communication)

Scatter and Gather (1)

Scatter: one node

unique message to each of the nodes

one-to-all personalized communication

Total # of messages $= p$

Gather:

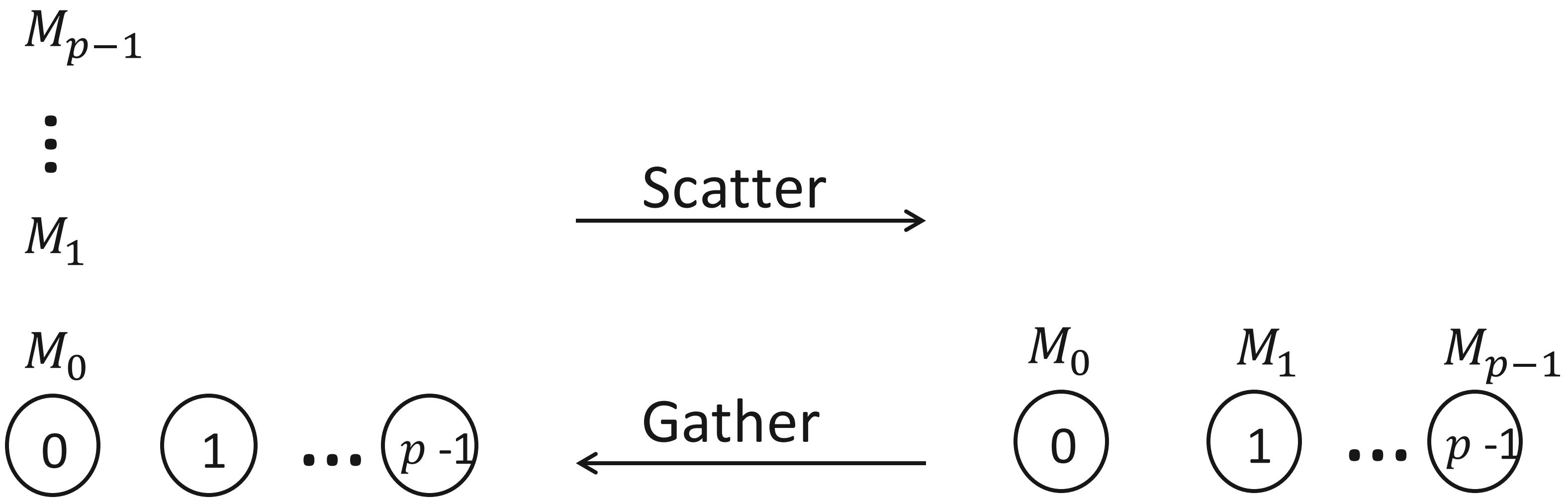
Each node has a unique message

Single node collects all messages

(No reduction)

Scatter and Gather (2)

Scatter and Gather operations



Performing Scatter (1)

Scatter on Hypercube

Idea of one-to-all broadcast can be used

Recursive doubling

P_0 has the p messages initially

$$A(0), \dots, A(p-1)$$

1. Send $A\left(\frac{p}{2}\right), \dots, A(p-1)$ to $P_{\frac{p}{2}}$
2. Recursively perform Scatter (Two Scatter operations of size $\frac{p}{2}$ in parallel)

Performing Scatter (2)

On hypercube (Network model)

$$T_p(p) = {}^{p}/_{2} + T_{{}^{p}/_{2}}({}^{p}/_{2})$$

$$T_2(2) = 1$$

$$T_p(p) = O(p)$$

Message size = ${}^{p}/_{2}$

Note: Single port communication

Overlap Step 1 and 2

Performing Scatter (3)

Linear Array (1-D Mesh) Network Model

Each message $=$ 1 unit

Send $A\left(\frac{p}{2}\right), \ldots, A(p-1)$ from P_0 to $P_{\frac{p}{2}}$

Pipeline

Time $= \left({}^{p}/_{2} - 1\right) + \left({}^{p}/_{2} - 1\right)$

Perform two Scatter operations

recursively on two linear arrays of size ${}^{p}/_{2}$

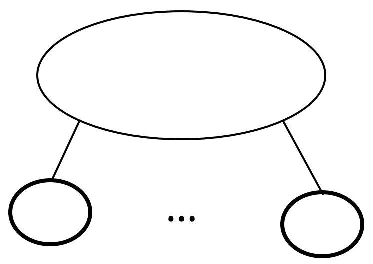
Parallel Algorithms using Communication Primitives

Application Developer

Communication Primitives
(eg. All-to-all Broadcast, …)

Basic communication Primitives
(eg. Send/Receive/Barrier)

Hardware →



Example: Dense Matrix Vector Multiplication (1)

$C \leftarrow A \times B$

$C: n \times 1$ vector

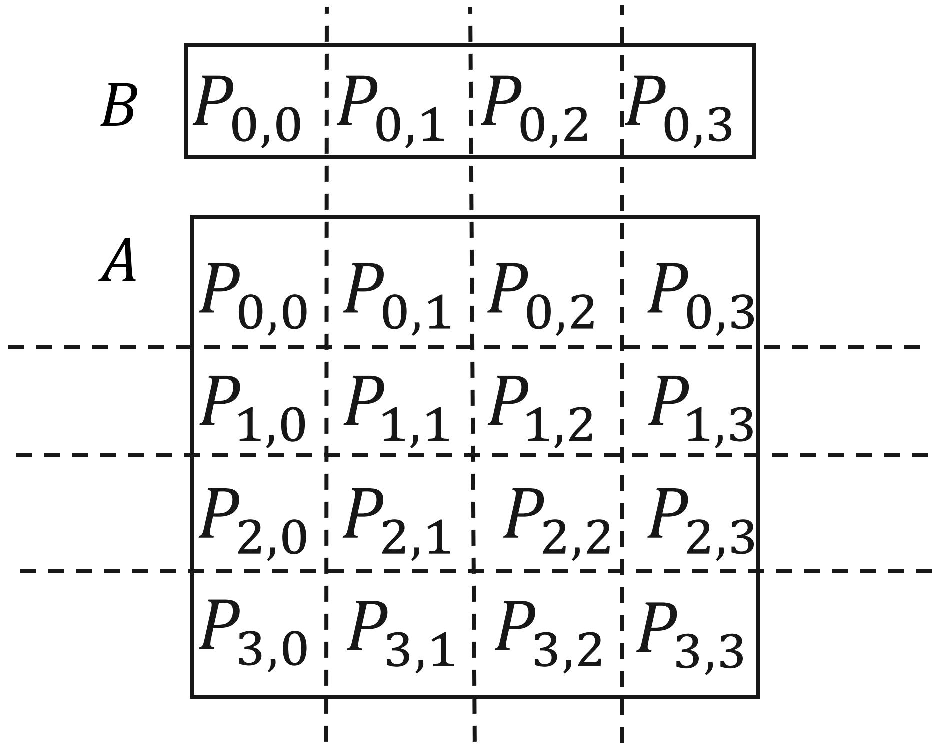
$A: n \times n$ matrix

$B: n \times 1$ vector

n^2 Processors in $n \times n$ mesh

Initial data alignment

- $P_{i,j}$ has $A_{i,j}$ $(0 \le i, j < n)$
- $P_{0,j}$ has B_j $(0 \le j < n)$ [Row 0 has B]



$n \times n$ mesh for $n = 4$

Example: Dense Matrix Vector Multiplication (2)

Step 1:

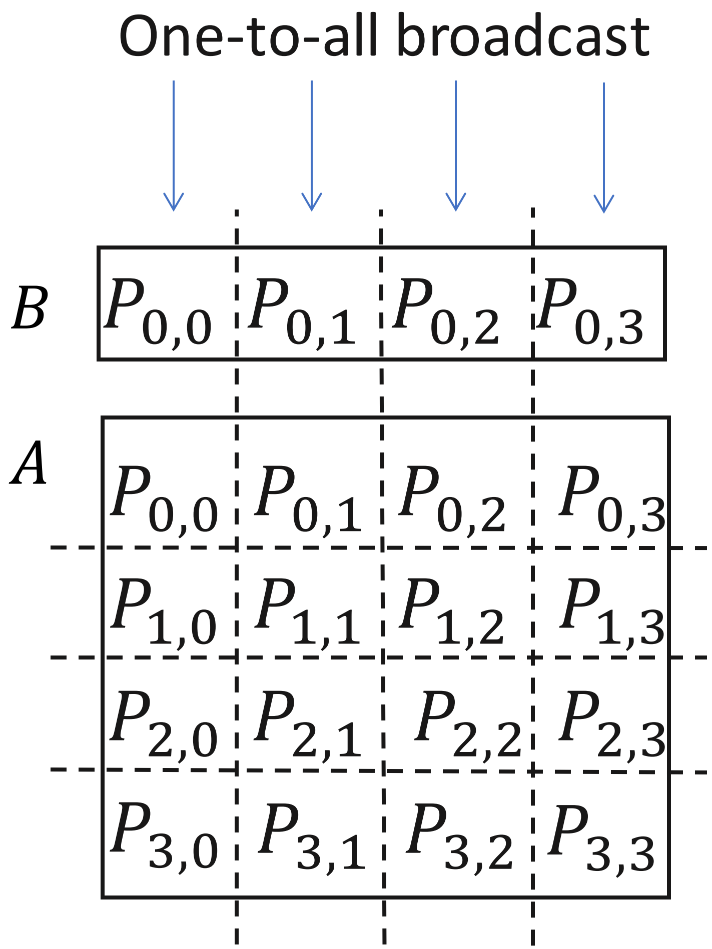
One-to-all broadcast along each column

(n concurrent broadcasts)

Do j = 0 to $n - 1$ in parallel

$P_{0,j}$ broadcasts B_j to $P_{1,j}, \dots, P_{n-1,j}$

End



Example: Dense Matrix Vector Multiplication (3)

Step 2:

Perform local computation in each processor

Do i = 0 to $n-1$ in parallel

Do j = 0 to $n-1$ in parallel

$P_{i,j}$ computes $A_{i,j} \times B_j$

End

End

Time $= O(1)$

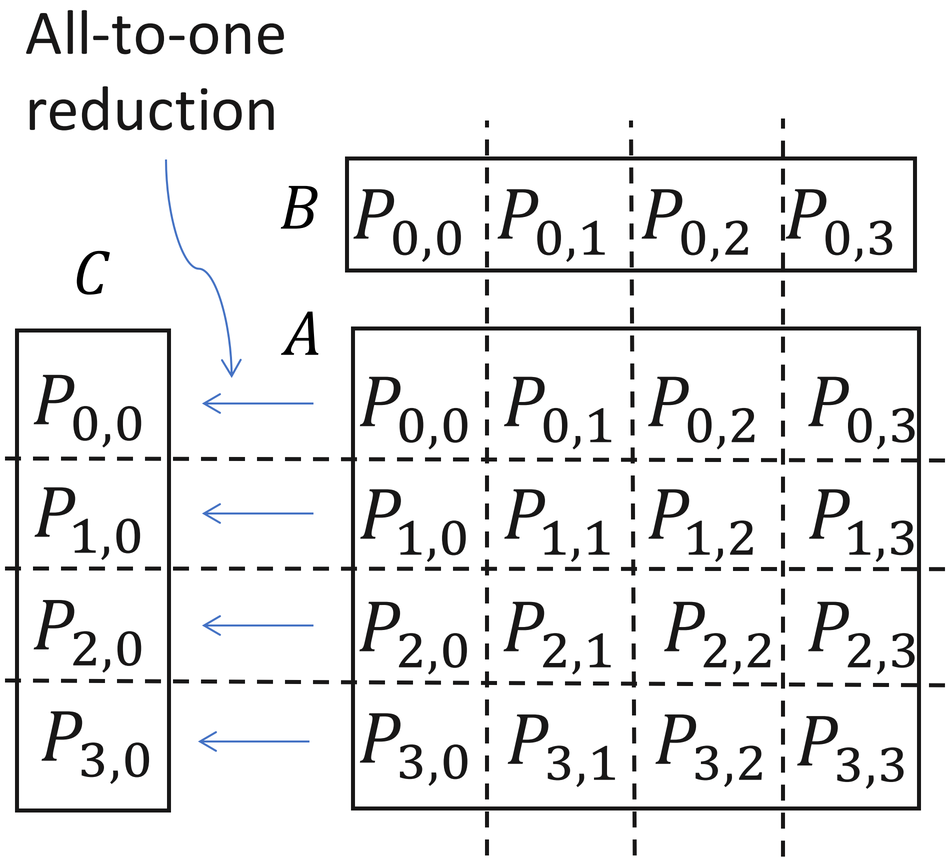
Example: Dense Matrix Vector Multiplication (4)

Step 3:
All-to-one Reduction in each row

Do i = 0 to $n-1$ in parallel

$$C_i = \sum_{j=0}^{n-1} A_{i,j} \times B_j$$

End



Example: Dense Matrix Vector Multiplication (5)

Analysis using Network model ($n \times n$ mesh)

Total time $= (n-1)$	Broadcast (Step 1)
$+1$	Compute (Step 2)
$+(n-1)$	Reduction (Step 3)

Total time $= O(n)$

Example: Matrix Multiplication (1)

$C \leftarrow A \times B$

$n \times n$ matrices

$n \times n$ Processors in 2D mesh

Initial data alignment

- $P_{i,j}$ has $A_{i,j}$ and $B_{i,j}$

$P_{0,0}$	$P_{0,1}$	$P_{0,2}$	$P_{0,3}$
$P_{1,0}$	$P_{1,1}$	$P_{1,2}$	$P_{1,3}$
$P_{2,0}$	$P_{2,1}$	$P_{2,2}$	$P_{2,3}$
$P_{3,0}$	$P_{3,1}$	$P_{3,2}$	$P_{3,3}$

$n \times n$ mesh for $n = 4$

Example: Matrix Multiplication (2)

$C(i,j) \leftarrow 0$

Do $k = 1$ to n

Broadcast Row_k of B along columns

$[B_{k,j}\ (1 \le j \le n)]$

Broadcast Col_k of A along rows

$C(i,j) \leftarrow C(i,j) + A(i,k) \times B(k,j)$

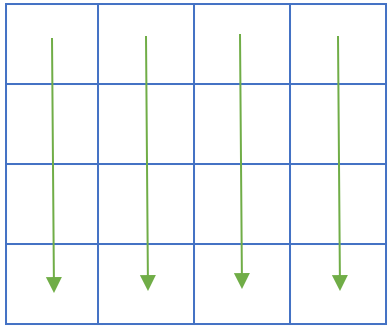
End

$2n$ broadcast operations in parallel

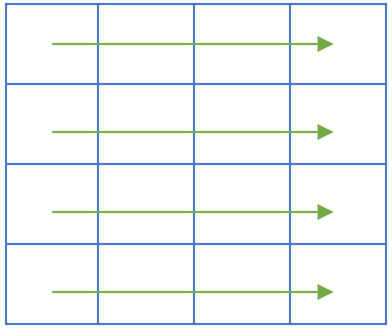
n^2 operations in parallel

$P_{i,j}$ has $A_{i,k}$ and $B_{k,j}$ in k^{th} iteration

B Matrix



A Matrix



$k = 1$

Example: Matrix Multiplication (3)

Analysis using Network model ($n \times n$ mesh)

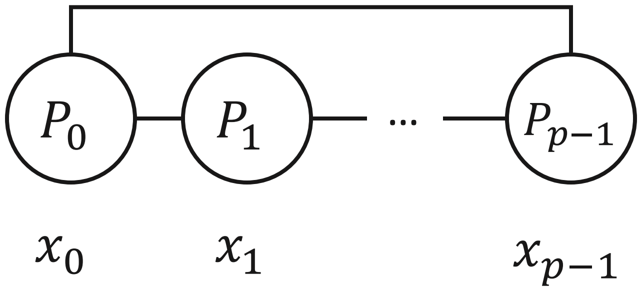
Total time $= [(n-1)$	Broadcast
$+1$	Compute
$] \times n$	n iterations

Total time $= O(n^2)$

Not work optimal

Note: By overlapping broadcast operations (iterations), total time can be improved to $O(n)$

Example: Sorting on a Linear Array using Communication Primitives (1)



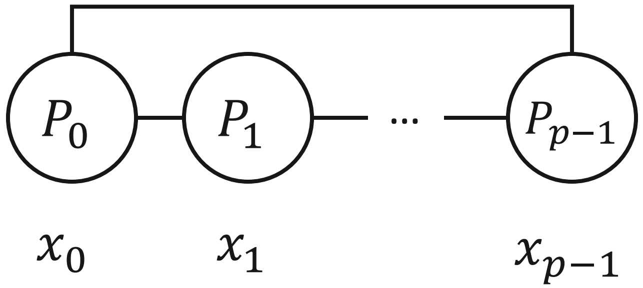
p elements $x_0, \ldots, x_{p-1}$

p processors in a linear array with wraparound

Initial data alignment: P_i has x_i $(0 \leq i < p)$

Compute rank (x_i) = # of elements < x_i

Example: Sorting on a Linear Array using Communication Primitives (2)



In each processor P_i

Rank(x_i) $\leftarrow$ 0

$M_{sent} \leftarrow x_i$

Do k = 1 to p

    Send M_{sent} to $P_{(i+1)\%p}$ Rotate right

    Receive $M_{received}$ from $P_{(i-1)\%p}$

    If ($M_{received} < x_i$) then

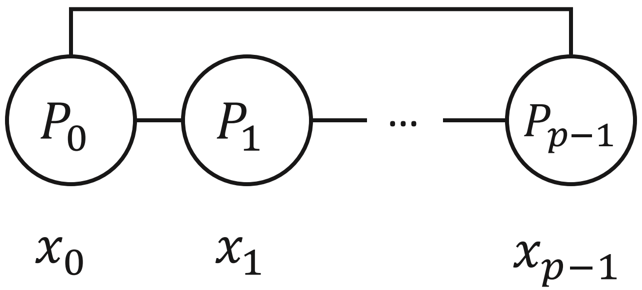
        Rank(x_i) $\leftarrow$ Rank(x_i) + 1

    End

    $M_{sent} \leftarrow M_{received}$

End

Example: Sorting on a Linear Array using Communication Primitives (3)



Compute Rank

Permute: Send x_i to Rank(x_i)

Total time = $O(p)$

Example: K-Means Clustering (1)

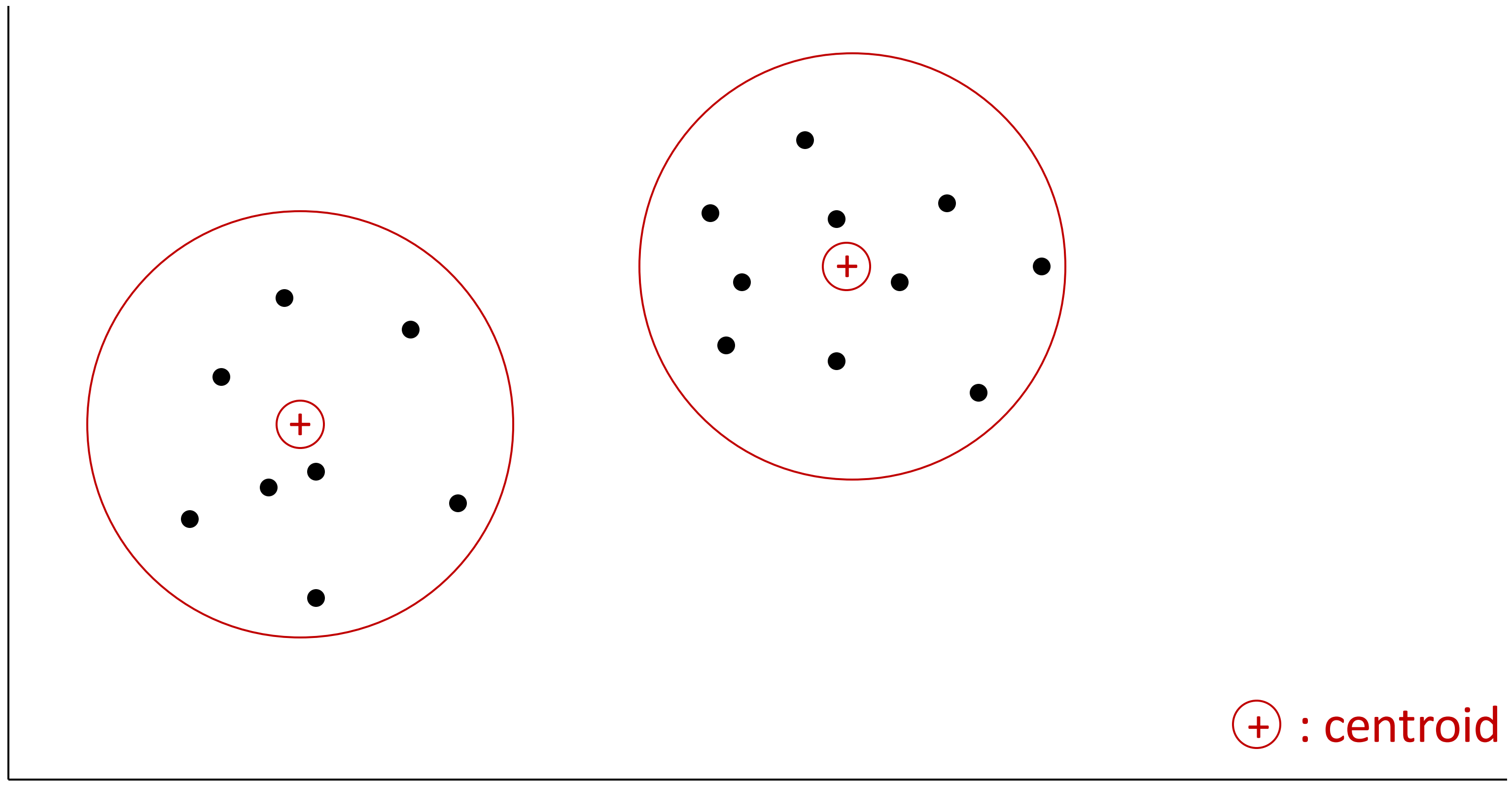
N data points $x_0, \ldots, x_{N-1}$ where $x_i \in R^2$

k clusters

Input: $x_0, \ldots, x_{N-1}$ and k

Objective: Assign N data points to k clusters such that the distance between each data point and the cluster centroid to which it is assigned is minimized.

Example: K-Means Clustering (2)



$$k = 2$$

Example: K-Means Clustering (3)

- **Unsupervised learning** is a type of machine learning algorithm used to draw inferences from datasets consisting of input data without labeled responses. The most common unsupervised learning method is cluster analysis.

- Common clustering algorithms include:
  - **Hierarchical clustering**: builds a multilevel hierarchy of clusters by creating a cluster tree
  - **k-Means clustering**: partitions data into k distinct clusters based on distance to the centroid of a cluster

Example: K-Means Clustering (4)

Centroid of a cluster

- Given points $x_0 = (a_0, b_0),\ x_1 = (a_1, b_1), \dots, x_l = (a_l, b_l)$ in a cluster:
- Centroid = (Average of x-axis, Average of y-axis)
- Centroid c = $\sum_i \frac{x_i}{l}$ = $(\sum_i \frac{a_i}{l}, \sum_i \frac{b_i}{l})$ Equation 1

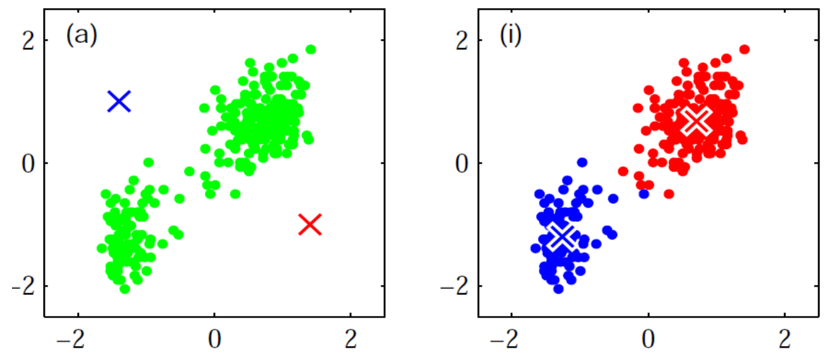
Example: K-Means Clustering (5)

Serial Program:

- Initialize $c_0, \dots, c_{k-1}$

Repeat:

- Initialize number of points assigned to cluster i: n_i $\forall i$ to 0.
- In each Iteration
  1. (Clustering) For each data point:
     - Compute its 'distance' to each c_i $\forall i \in \{0, \dots, k-1\}$.
     - Assign it to the cluster j with the closest centroid.
     - $n_j \leftarrow n_j + 1$
  2. (Centroid) Recompute
     - c_i $\forall i \in \{0, \dots, k-1\}$
     - using Equation 1



Example: K-Means Clustering (6)

- Total work (number of operations) per iteration:
  - Clustering (step 1): $O(Nk)$
  - Centroid recomputation (step 2): $O(Nk)$
  - Overall: $O(Nk)$ operations

Example: K-Means Clustering (7)

Parallel Algorithm for K-Means using Communication Primitives

N processors for N points + 1 processor (P_0) for centroids

Step 1 **Broadcast** k centroids to all processors

Step 2 **Local Computation**: Calculate distance from k clusters for the point and assign to the cluster with minimum distance

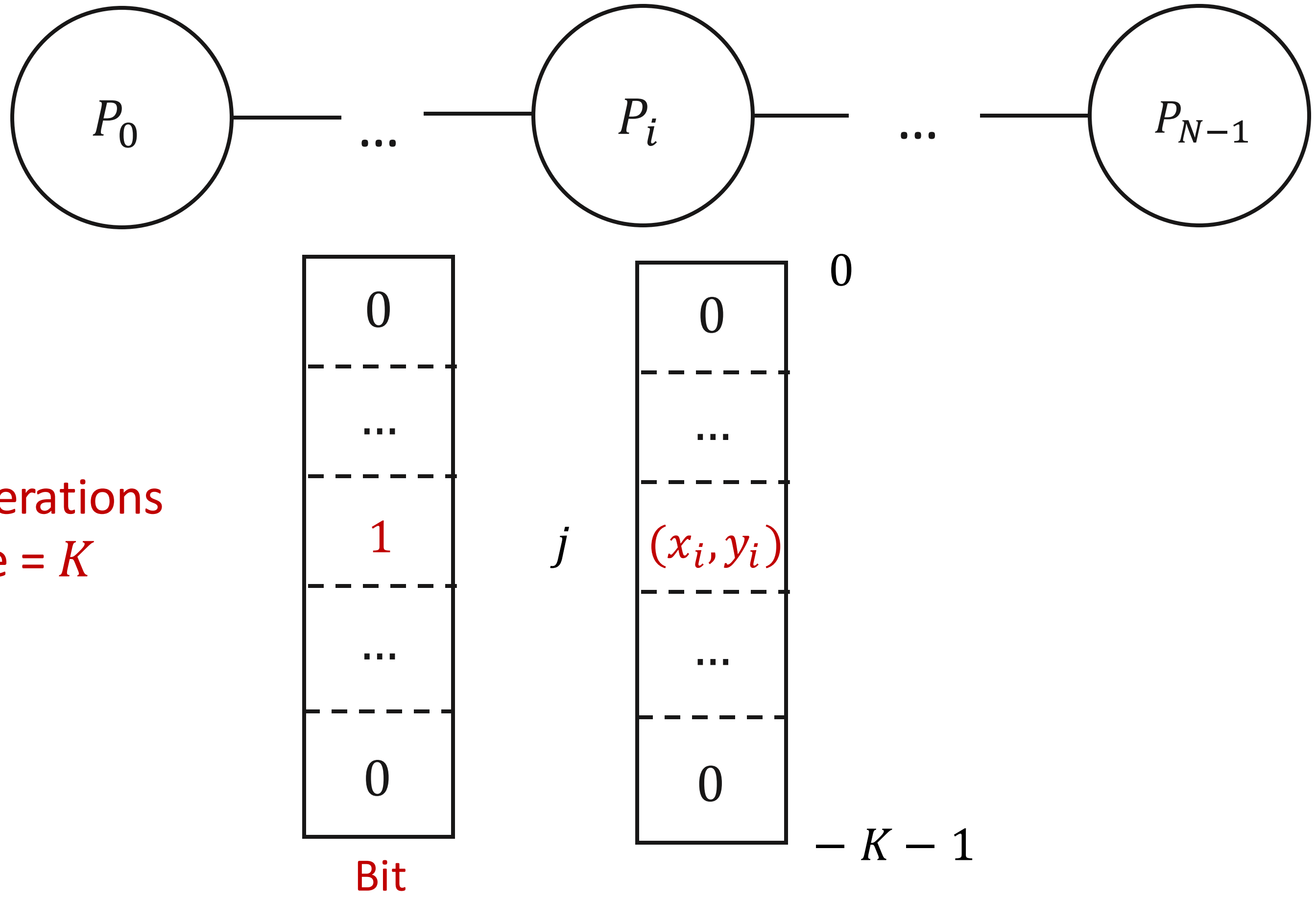
Step 3 **Reduction**: All-to-one reduce new centroids for each point to processor P_0

Step 4 **Centroid Update**: Processor P_0 updates all centroids

Example: K-Means Clustering (8)

Data structure for re-computing centroids

Processors:



2 Reduce operations
Message size = K

Example: K-Means Clustering (9)

Data structure for re-computing centroids

If (x_i, y_i) is assigned to j cluster, then set j^{th} entry $= (x_i, y_i)$

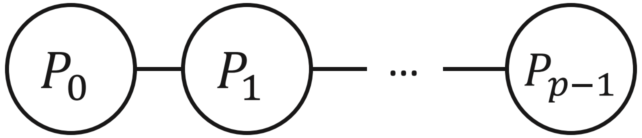
Number of 1 in the j^{th} position over all the processors = number of inputs assigned to cluster j

j^{th} bit $= 1$ if (x_i, y_i) assigned to cluster j

Example: K-Means Clustering (8)

K-means using communication primitives

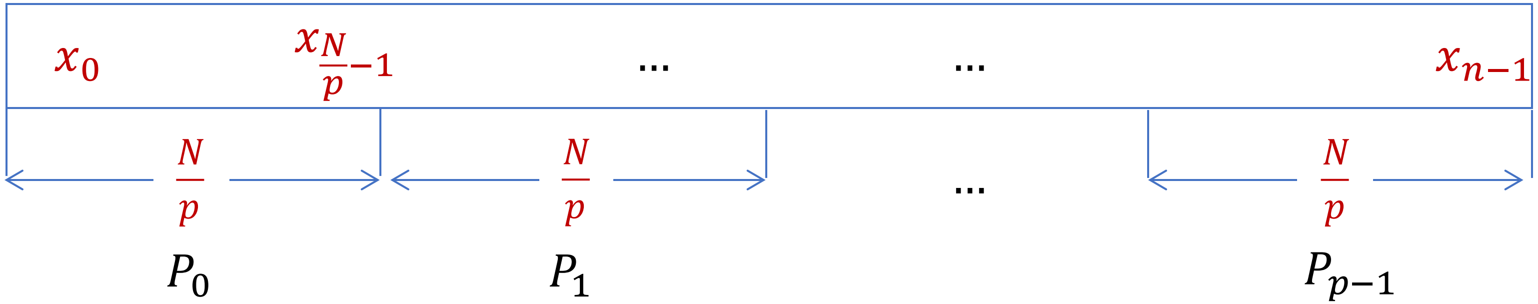
p processors $P_0, \ldots, P_{p-1}$ in a linear array



P_0 has the centroid values

Assign data as follows:

$$P_i \text{ has } x_{i\times\frac{N}{p}}, \ldots, x_{(i+1)\times\frac{N}{p}-1} \ (0 \le i < p) \quad \ldots$$



Block distribution

Example: K-Means Clustering (9)

K-means using communication primitives

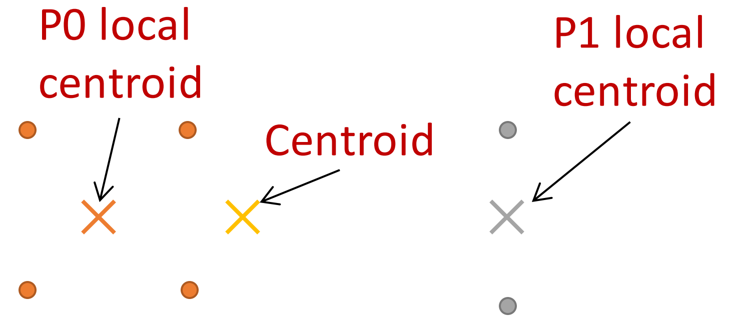
Each process P_i creates local clusters using data assigned to it

Local variables in Process P_i:

Local cluster centroids: $c_0^i, \ldots, c_{k-1}^i$

Local cluster sizes: $n_0^i, \ldots, n_{k-1}^i$

centroid-size product: $w_0^i, \ldots, w_{k-1}^i$



Master Process P_0:

Update centroid values using cluster centroids and sizes from all processes

centroid value = weighted mean of cluster centroids from all processes with cluster sizes as weights

$$c_j = \frac{\sum_{i=0}^{p-1} w_j^i}{\sum_{i=0}^{p-1} n_j^i}, \text{ where } w_j^i = n_j^i . c_j^i \qquad \text{Equation 2}$$

Example: K-Means Clustering (10)

In each iteration

1. P_0 broadcasts $c_0, \dots, c_{k-1}$ to $P_1, \dots, P_{p-1}$
2. In each processor P_i: do in parallel

    $c_0^i, \dots, c_{k-1}^i \leftarrow c_0, \dots, c_{k-1}$
    $n_0^i, \dots, n_{k-1}^i \leftarrow 0,0, \dots, 0$
    $w_0^i, \dots, w_{k-1}^i \leftarrow 0,0, \dots, 0$
    For each data point assigned to it

        Compute its 'distance' to each c_i $\forall i \in \{0, \dots, k-1\}$.
        Assign it to the cluster j with minimum 'distance'
        $n_j \leftarrow n_j + 1$

    Update c_j^i $\forall j \in \{0, \dots, k-1\}$
    Assign $w_j^i = n_j c_j^i$ $\forall j \in \{0, \dots, k-1\}$

3. All-to-one reduce $w_0^i, \dots, w_{k-1}^i$ to P_0 all $i \in \{0,1, \dots, p-1\}$
4. All-to-one reduce $n_0^i, \dots, n_{k-1}^i$ to P_0 all $i \in \{0,1, \dots, p-1\}$
5. P_0 updates c_j for all $j \in \{0,1, \dots, k-1\}$ using Equation 2

Example: K-Means Clustering (11)

All to one Reduction

P_0		P_0	P_1		P_{p-1}
w_0	$\leftarrow$	w_0^0	w_0^1	…	w_0^{p-1}
w_1	$\leftarrow$	w_1^0	w_1^1	…	w_1^{p-1}
…	$\leftarrow$	…	…	…	…
w_{k-1}	$\leftarrow$	w_{k-1}^0	w_{k-1}^1	…	w_{k-1}^{p-1}

P_0		P_0	P_1		P_{p-1}
n_0	$\leftarrow$	n_0^0	n_0^1	…	n_0^{p-1}
n_1	$\leftarrow$	n_1^0	n_1^1	…	n_1^{p-1}
…	$\leftarrow$	…	…	…	…
n_{k-1}	$\leftarrow$	n_{k-1}^0	n_{k-1}^1	…	n_{k-1}^{p-1}

Example: K-Means Clustering (12)

Time per iteration

Message size: k (number of clusters)

Step 1 (Broadcast): k messages

Step 2 (Local Computation): Calculate distance from k clusters for $\frac{N}{p}$ points and assign to cluster with minimum distance

Step 3, 4 (Reduction): k messages

Step 5 (Centroid update): k centroid updates

Assume no pipelining for broadcast and reduction

Example: K-Means Clustering (13)

Analysis using Network model

Total time per iteration $= O(p \cdot k)$	Broadcast
$+ O(\frac{N}{p} \cdot k)$	Local computation
$+ O(p \cdot k)$	Reduction
$+ O(k)$	Centroid Update

Summary

Implementation of communication primitives

- One-to-all Broadcast in a 2-D Mesh
- One-to-all Broadcast in a Hypercube
- All-to-all Broadcast
- Scatter and Gather

Examples

- Dense Matrix Vector Multiplication
- Matrix Multiplication
- Sorting on a linear array
- K-means clustering