# Parallel and Distributed Computing
# CS3006 (BDS-6A)
# Lecture 16

Instructor: Dr. Syed Mohammad Irteza

Assistant Professor, Department of Computer Science, FAST

28 March, 2023

# Previous Lecture

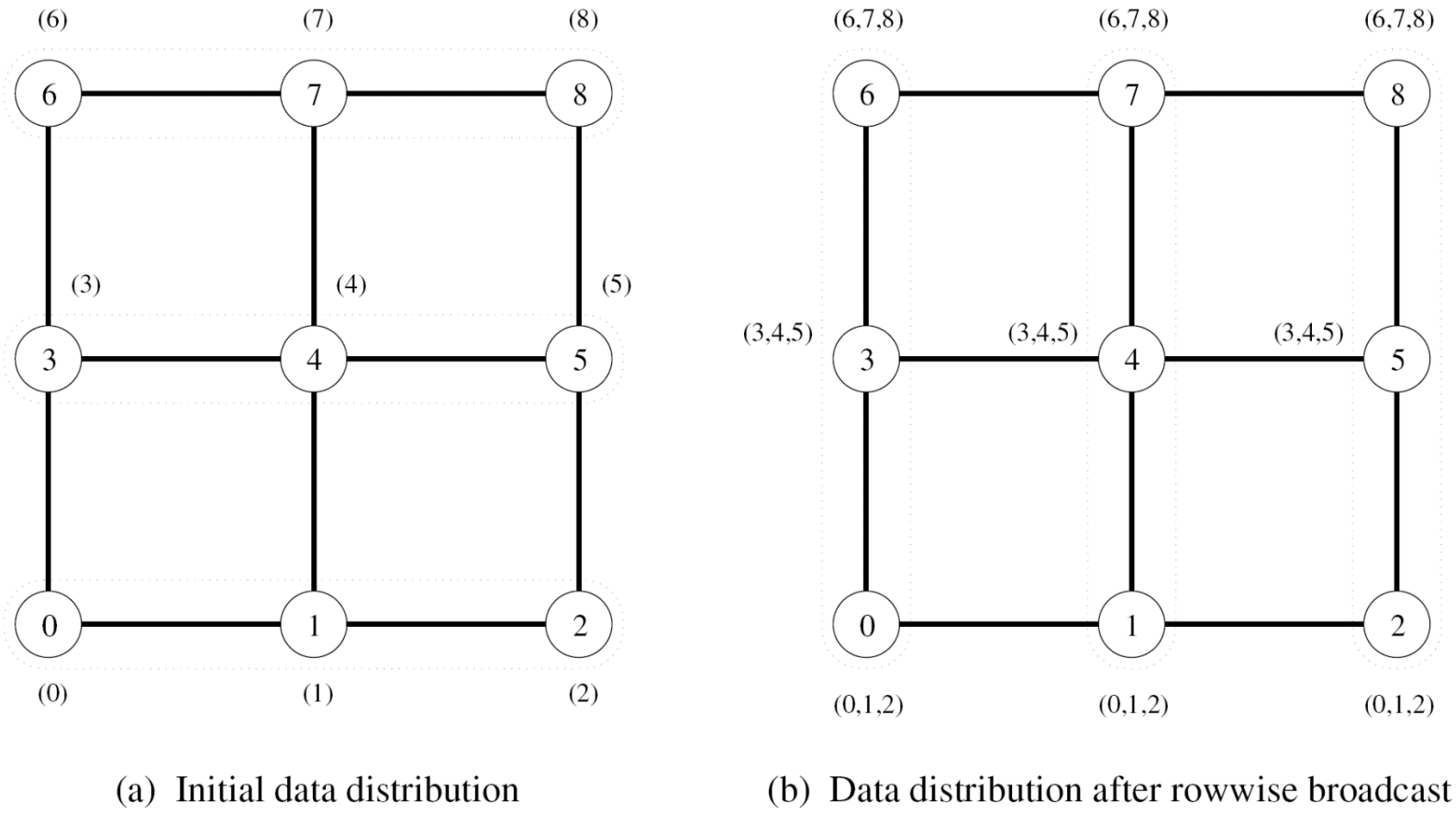
- Basic Communication Operations
  - Recap: 1-to-all Broadcast, All-to-1 Reduction
    - Under linear array, ring, mesh, hypercube
    - Using naïve method and recursive doubling
  - All-to-All Broadcast
    - Over Ring & mesh
  - All-to-All Reduction
    - Over Ring & mesh
  - Cost Estimates

# Linear Ring Reduction

```
1.  procedure ALL_TO_ALL_RED_RING(my_id, my_msg, p, result)
2.  begin
3.      left := (my_id − 1) mod p;
4.      right := (my_id + 1) mod p;
5.      recv := 0;
6.      for i := 1 to p − 1 do
7.          j := (my_id + i) mod p;
8.          temp := msg[j] + recv;
9.          send temp to left;
10.         receive recv from right;
11.     endfor;
12.     result := msg[my_id] + recv;
13. end ALL_TO_ALL_RED_RING
```

**Algorithm 4.5** All-to-all reduction on a $p$-node ring.

# All-to-All Broadcast on 2D Mesh




(a) Initial data distribution

(b) Data distribution after rowwise broadcast

**Figure 4.10** All-to-all broadcast on a $3 \times 3$ mesh. The groups of nodes communicating with each other in each phase are enclosed by dotted boundaries. By the end of the second phase, all nodes get (0,1,2,3,4,5,6,7) (that is, a message from each node).

# All-to-All Broadcast on 2D Mesh Algorithm

```
1.    procedure ALL_TO_ALL_BC_MESH(my_id, my_msg, p, result)
2.    begin

/* Communication along rows */
3.        left := my_id − (my_id mod √p) + (my_id − 1) mod √p;
4.        right := my_id − (my_id mod √p) + (my_id + 1) mod √p;
5.        result := my_msg;
6.        msg := result;
7.        for i := 1 to √p − 1 do
8.            send msg to right;
9.            receive msg from left;
10.           result := result ∪ msg;
11.       endfor;

/* Communication along columns */
12.       up := (my_id − √p) mod p;
13.       down := (my_id + √p) mod p;
14.       msg := result;
15.       for i := 1 to √p − 1 do
16.           send msg to down;
17.           receive msg from up;
18.           result := result ∪ msg;
19.       endfor;
20.   end ALL_TO_ALL_BC_MESH
```

**Algorithm 4.6** All-to-all broadcast on a square mesh of $p$ nodes.

# All-to-All Broadcast on Hypercube

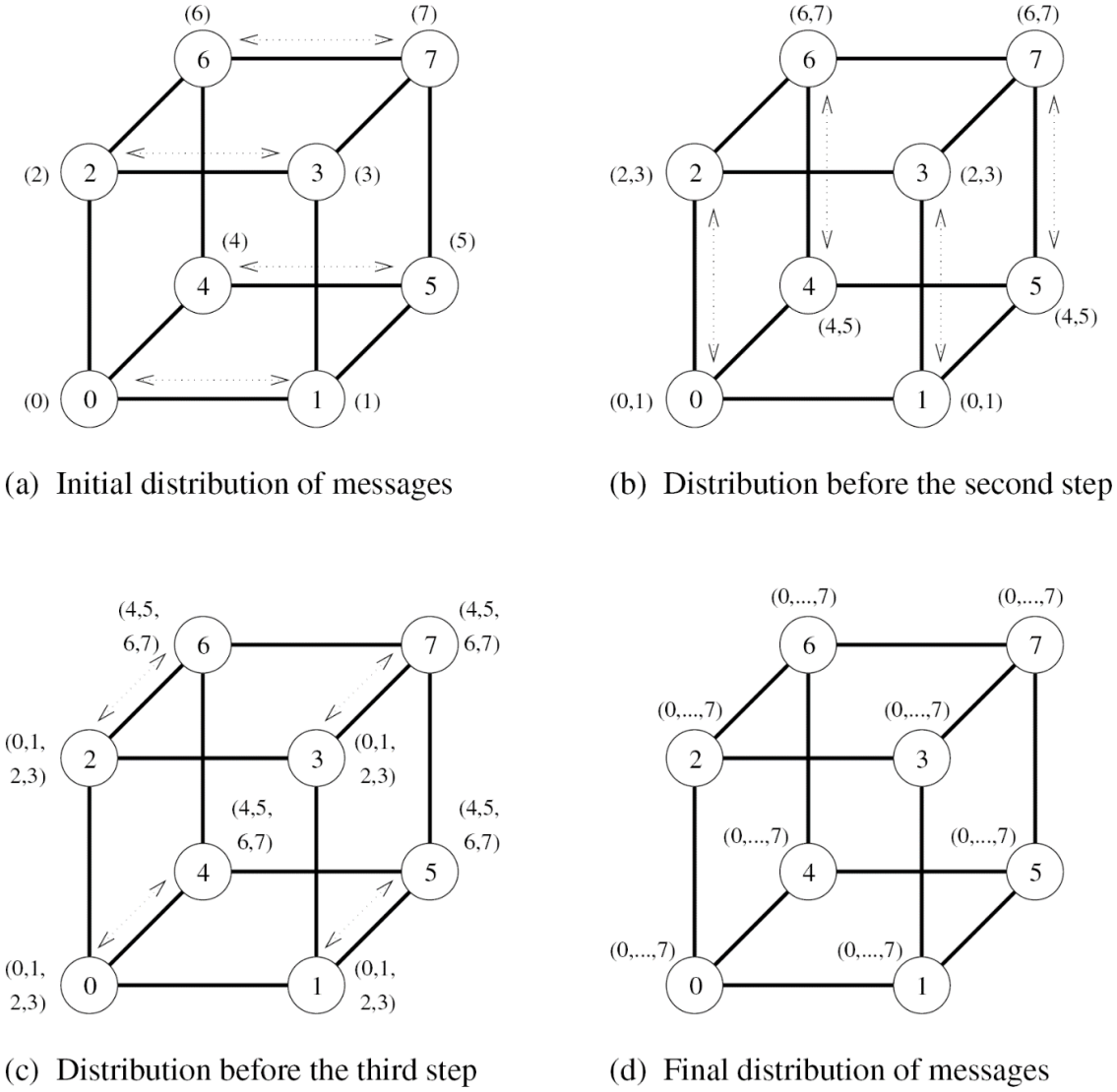



**Figure 4.11** All-to-all broadcast on an eight-node hypercube.

# All-to-All Broadcast on Hypercube. Algorithm

```
1.   procedure ALL_TO_ALL_BC_HCUBE(my_id, my_msg, d, result)
2.   begin
3.       result := my_msg;
4.       for i := 0 to d − 1 do
5.           partner := my_id XOR 2^i;
6.           send result to partner;
7.           receive msg from partner;
8.           result := result ∪ msg;
9.       endfor;
10.  end ALL_TO_ALL_BC_HCUBE
```

**Algorithm 4.7** All-to-all broadcast on a $d$-dimensional hypercube.

# All-to-All Broadcast and All-to-All Reduction

**Cost Estimation**

- Different on each infrastructure
- **Linear Ring**
  - $\boldsymbol{T = (t_s + mt_w)(P - 1)}$

- **Mesh**
  - Total time for All-to-All broadcast in the first phase
    - $\boldsymbol{T(first\ phase) = (t_s + mt_w)(\sqrt{p} - 1)}$
  - Total time for the second phase (note here m= $\sqrt{\boldsymbol{p}}$.m)
    - $\boldsymbol{T(Second\ phase) = (t_s + (\sqrt{p})mt_w)(\sqrt{p} - 1)}$
- So, Total time= $\boldsymbol{2t_s(\sqrt{p} - 1) + mt_w(p - 1)}$

# **Cost Estimation:** All-to-All Broadcast and All-to-All Reduction

- Different on each infrastructure.
- **Hypercube (broadcast)**
  - Communication in for 1st step: $(\boldsymbol{t_s}+\boldsymbol{mt_w})$
  - Communication in for 2nd step: $(\boldsymbol{t_s}+\boldsymbol{2mt_w})$
  - Communication in for ith step: $(\boldsymbol{t_s}+\boldsymbol{2^{i-1}mt_w})$
  - Total Cost= $\sum_{i=1}^{\log(p)}(t_s+2^{i-1}mt_w)$

**Cost Estimation:** All-to-All Broadcast and All-to-All Reduction

- Total Cost= $\sum_{i=1}^{\log(p)}(t_s+2^{i-1}mt_w)$
- Simplify the equation
- HINT: [$x^0 + x^1$ + ……+ $x^n = \frac{x^{n+1}-1}{x-1}$ ]

- Answer $\boldsymbol{T = (t_s \log p + mt_w(p-1))}$

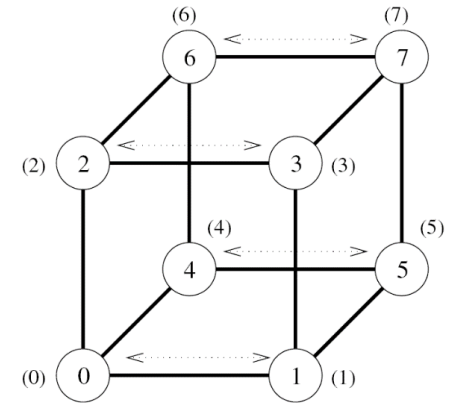
# All-Reduce

# All-Reduce

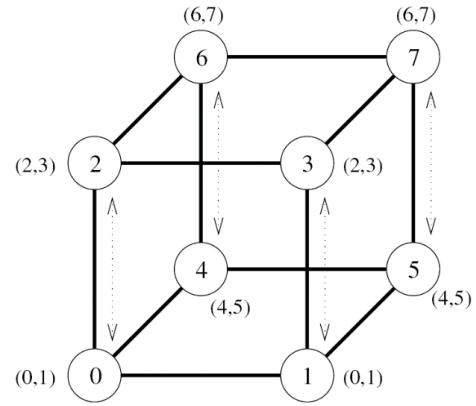
- Precondition: Every process *i* has a single message $M_i$ of size *m words.*
- Post condition: All processes have a reduced message *M of size m words.*

**Strategies:**

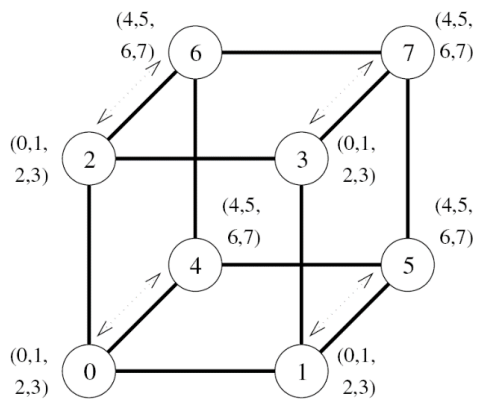
1. Use **all-to-one reduction** followed by **one-to-all** broadcast ($\boldsymbol{2 * (t_s + mt_w)} \log \boldsymbol{p}$)
2. Use **modified All-to-All comm.** algorithm for hypercube ($(\boldsymbol{t_s + mt_w}) \log \boldsymbol{p}$)
   - Replace Union with associative operator



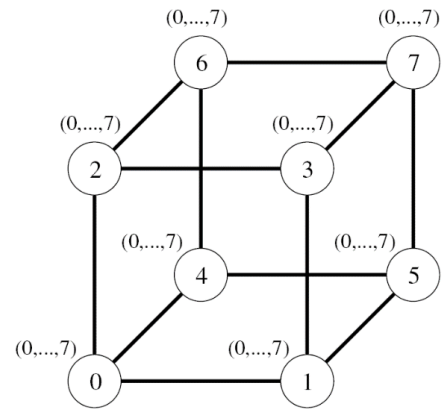

(a) Initial distribution of messages




(b) Distribution before the second step




(c) Distribution before the third step




(d) Final distribution of messages

**Figure 4.11** All-to-all broadcast on an eight-node hypercube.

# Prefix-Sum

# Prefix-Sums

- Prefix-sums are also known as scan operations
- Given $p$ numbers $n_0, n_1, \ldots, n_{p-1}$ (one on each node), the problem is to compute the sums such that: -
  - $S_k = \sum_{i=0}^{K}(n_i)$
    - Here $S_k$ is the prefix-sum computed at kth node after the operation.

  **Example:**
  - Original sequence: <3, 1, 4, 0, 2>
  - Sequence of prefix sums: <3, 4, 8, 8, 10>

# Prefix-Sums

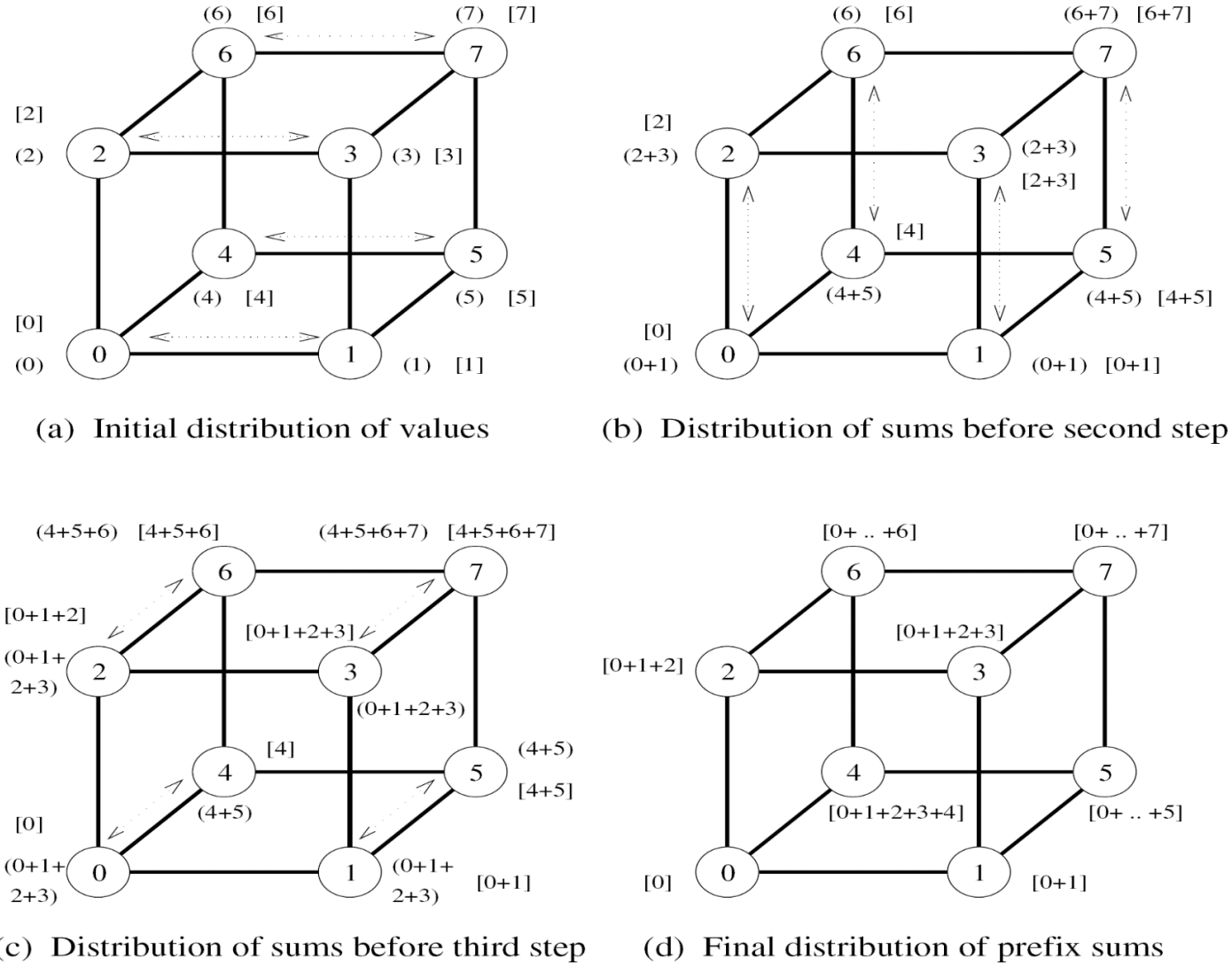



**Figure 4.13** Computing prefix sums on an eight-node hypercube. At each node, square brackets show the local prefix sum accumulated in the result buffer and parentheses enclose the contents of the outgoing message buffer for the next step.

# Prefix-Sums

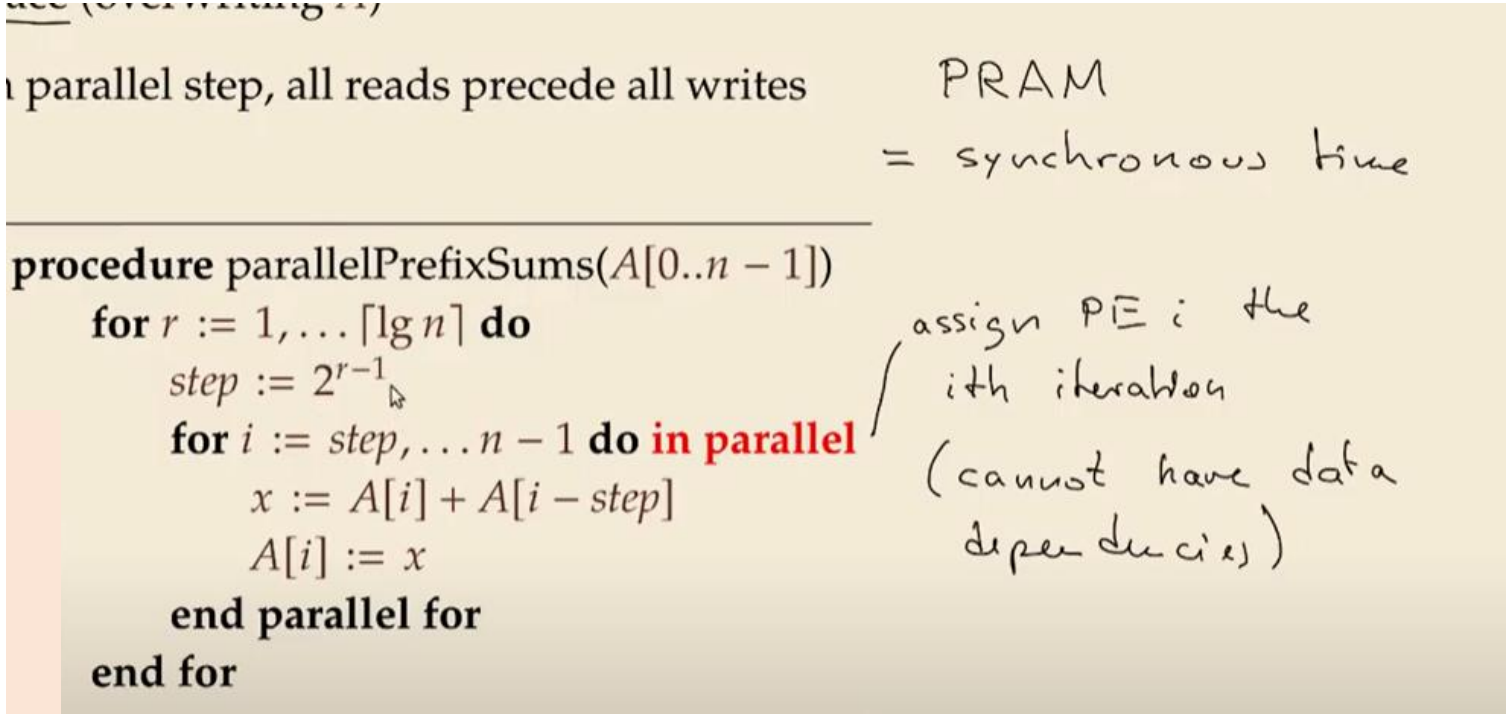
```
1.   procedure PREFIX_SUMS_HCUBE(my_id, my_number, d, result)
2.   begin
3.       result := my_number;
4.       msg := result;
5.       for i := 0 to d − 1 do
6.           partner := my_id XOR 2^i;
7.           send msg to partner;
8.           receive number from partner;
9.           msg := msg + number;
10.          if (partner < my_id) then result := result + number;
11.      endfor;
12.  end PREFIX_SUMS_HCUBE
```

**Algorithm 4.9** Prefix sums on a $d$-dimensional hypercube.

# COMP526 3-7 §3.6 Parallel primitives, Prefix sum - YouTube

| 3 | 0 | 0 | 5 | 7 | 0 | 0 | 2 | 0 | 0 | 0 | 4 | 0 | 8 | 0 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

| 3 | 3 | 3 | 8 | 15 | 15 | 15 | 17 | 17 | 17 | 17 | 21 | 21 | 29 | 29 | 30 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|



# References


1. Slides from Dr. Rana Asif Rehman & Dr. Haroon Mahmood
2. Kumar, V., Grama, A., Gupta, A., & Karypis, G. (1994). *Introduction to parallel computing* (Vol. 110). Redwood City, CA: Benjamin/Cummings.
3. Quinn, M. J. Parallel Programming in C with MPI and OpenMP,(2003).


Useful Links:

COMP526 3-7 §3.6 Parallel primitives, Prefix sum - YouTube