Parallel Algorithms

by

Biing-Feng Wang
王 炳 豐
Department of Computer Science
National Tsing Hua University

References:

[1] J. JáJá, *An Introduction to Parallel Algorithms*, Addison Wesley, 1992.
[2] S. G. Akl, *Parallel Computation: Models and Methods*, Prentice-Hall, 1997.
[3] Journals and proceedings

An Example

Problem : Sorting (Odd-Even Transposition Sort)
Input : 7, 6, 2, 1, 3, 5, 4
Output : 1, 2, 3, 4, 5, 6, 7
Model : Linear Processor Array (n = 7)

○ processor
— link

7 — 6 — 2 — 1 — 3 — 5 — 4

Step	P_1	P_2	P_3	P_4	P_5	P_6	P_7
1	7	6	2	1	3	5	4
2	6	7	1	2	3	5	4
3	6	1	7	2	3	4	5
4	1	6	2	7	3	4	5
5	1	2	6	3	7	4	5
6	1	2	3	6	4	7	5
7	1	2	3	4	6	5	7
	1	2	3	4	5	6	7

algo. { correctness ???
time complexity ???

Theorem: Odd-Even transposition sort produces a sorted sequence of n data after n steps.

$O(n)$ time $\Omega(n\cancel{\lg}n)$ lower bound

Outline

Part I

1. Introduction
   - Classification of Parallel Computers
   - Performance of Parallel Algorithms
2. Shared-Memory Computers, Basic Techniques, and Brent's Theorem

Part II

3. Tree Machines
4. Linear Processor Arrays
5. Mesh-Connected Computers
6. Hypercubes
7. Perfect Shuffles
8. Mesh-Connected Computers with Multiple Broadcasting
9. Processor Arrays with Reconfigurable Bus Systems

Part III

10. Systolic Architectures
11. Randomized Algorithms and P-Completeness

Grading

- Midterm Examination 50%
- Final Examination 40%
- Image 10%

$\Rightarrow$ normalize

- Phone/Notebook -5%/each
- Fail if 3^+ absences

Office Hours: 14:30 ~ 15:30, Monday to Friday
(R548, EECS)

Webpage: eLearn

1-3a

1

Introduction

Classification of Parallel Computers

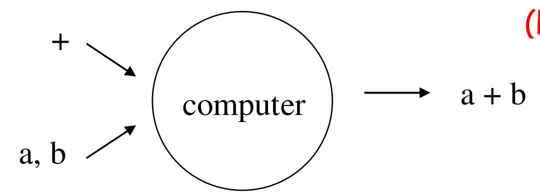
1. Flynn's Classification
   (M. J. Flynn, "Very high speed computing systems," *Proceedings of the IEEE*, vol. 54, 1966, pp. 1901-1909)

Instruction stream (*I*): a sequence of instructions performed by a computer

Data stream (*D*): a sequence of data used to execute an instruction stream

SISD: single instruction stream, single data stream

sequential computer
(PC with 1 CPU)

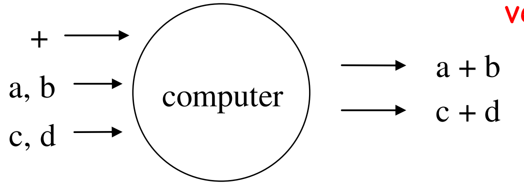


One instruction is performed at a time, on one set of data.

practical,
easy to implement

SIMD: single instruction stream, multiple data streams

vector computer
A + B

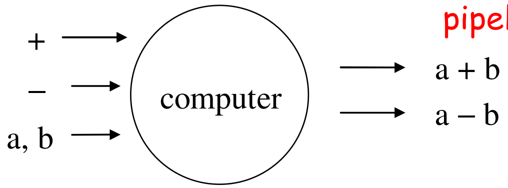


One type of instruction is performed at a time, possible on different sets of data.

MISD: multiple instruction streams, single data stream

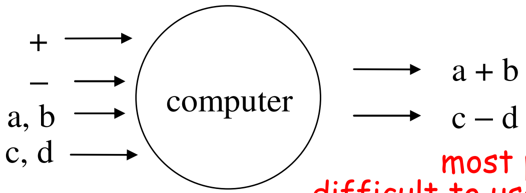
pipeline computer ??? ! !

1-5a



Different instructions on the same data can be performed at a time.

MIMD: multiple instruction streams, multiple data streams



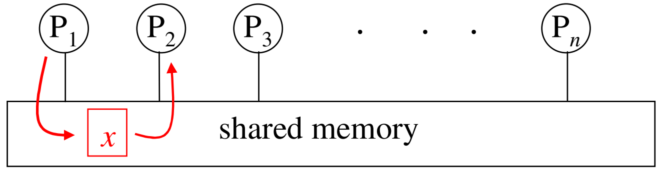
most powerful,
difficult to use (for many CPUs)

Different instructions on different data can be performed at a time.

1-5e 1-5f

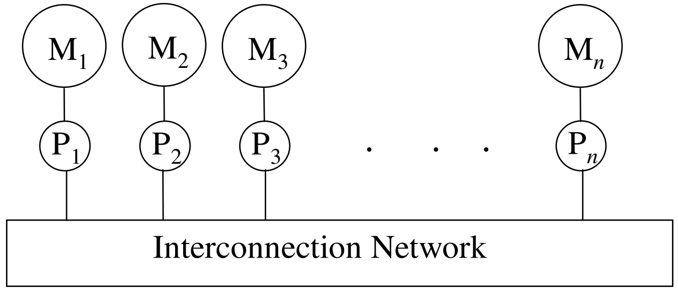
2. Schwartz's Classification
(J. T. Schwartz, "Ultra-computers", *ACM Transactions on Programming Languages and Systems*, vol. 2, no. 4, 1980, pp. 484-521.)

1. *Paracomputer* (*shared-memory computer*)



* Communication between any two processors takes $O(1)$ time through the shared memory
* The SIMD shared-memory computer is also called *Parallel Random Access Machine* (*PRAM*)

2. *Ultracomputer*



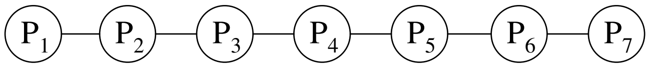
* The processors communicate with one another through an interconnection network. ⇒ one time unit (as fast as memory access)

≠ distributed systems (computer networks) (distributed algorithms)

Examples of widely used interconnection network:

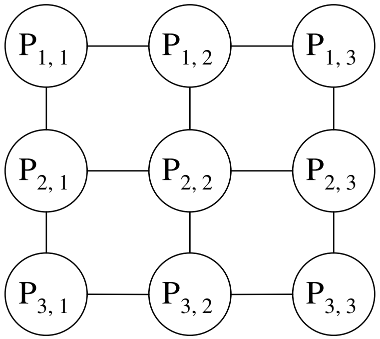
diameter
maximum degree
link complexity

1. Linear Processor Array (n = 7)



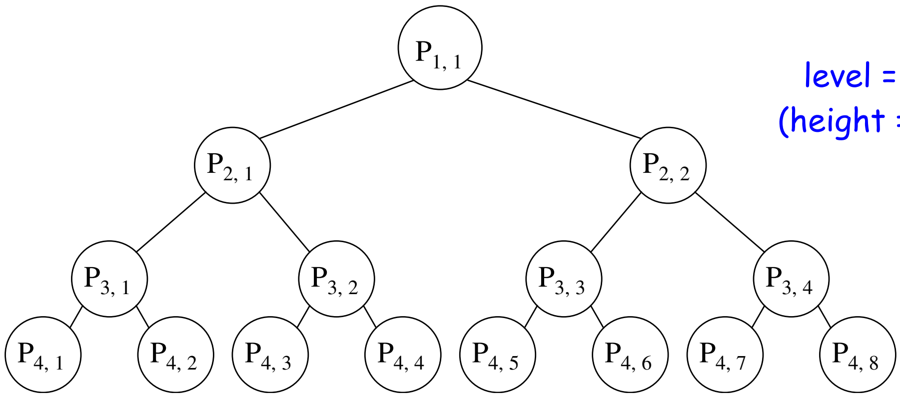
2. Mesh-Connected Computer (2d 3×3 mesh)

* k-d mesh: $d_1 \times d_2 \times \ldots \times d_k$
* 1-d mesh: LPA



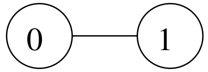
most practical
1st popular

3. Tree Machine ($n = 2^l - 1 = 2^4 - 1$, $l = \lg(n+1)$)



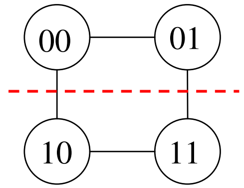
level = 4
(height = 3)

4. *Hypercube* (k-dimension ⇒ $n = 2^k$, $k = \lg n$)



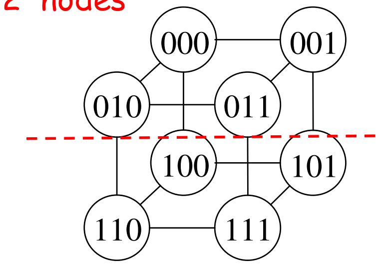
1-cube

2^1 nodes



2-cube

2^2 nodes



3-cube

2^3 nodes

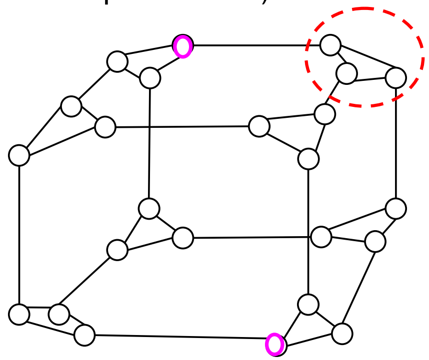
most important one
(most famous, the best, 10^{3+})

1-8a

5. *Cube-connected Cycle network* (each node of a k-cube is replayed by a cycle of k processors)

k-d ⇒ $n = 2^k \times k$

e.g., $k = 3$
$2^3 \times 3$ nodes

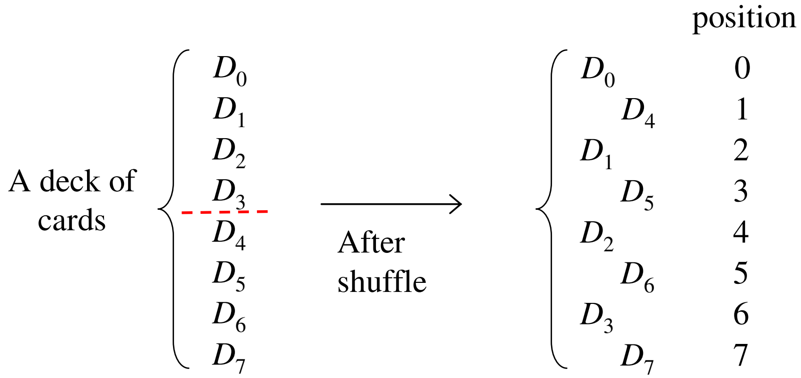


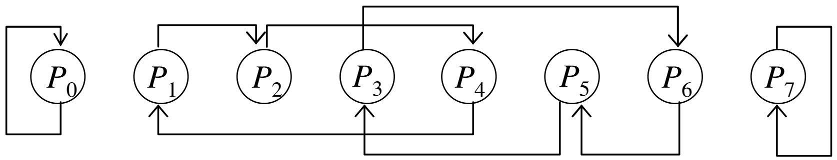
1-8b

diameter : = $O(kx(k/2)) = O(k^2)$
(= $2k - 2 + \lfloor k/2 \rfloor$ for k >= 4)

1-8d

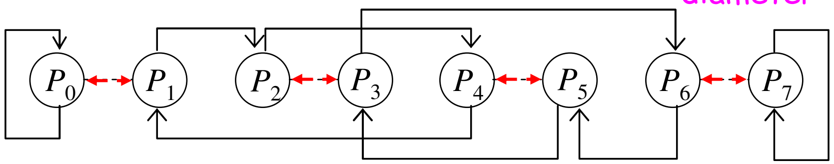
6. *Perfect Shuffle* k-d $\Rightarrow n = 2^k$ most interesting one



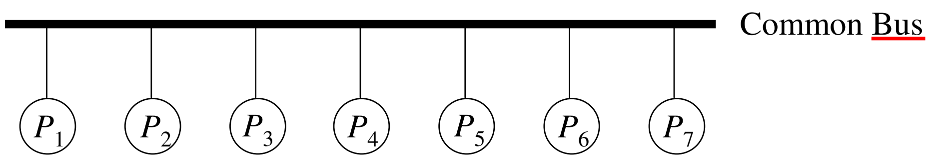


7. *Shuffle-Exchange Network*

diameter = $2k - 1$



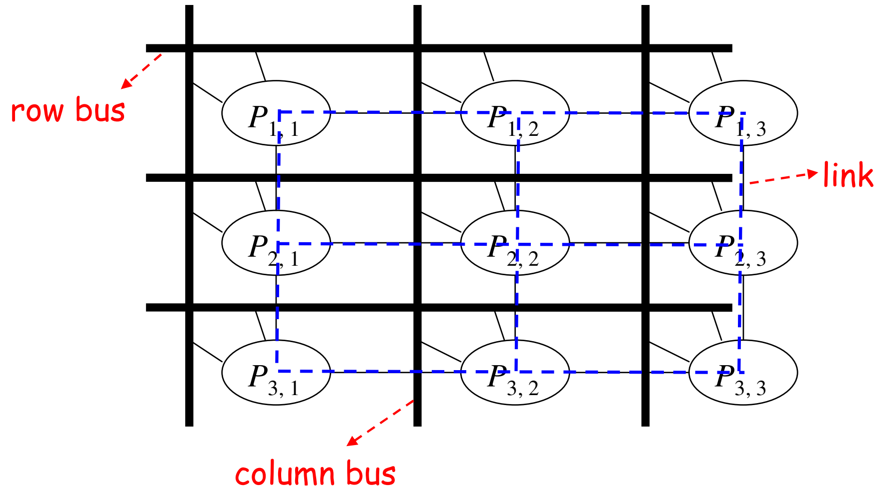
8. *Single-channel broadcast communication System*



* Broadcast a data requires $O(1)$ time.

pros: simple, $O(1)$ diameter
cons: one at a time

9. *Mesh-Connected Computer with Multiple Broadcasting*



*diameter???

* Current PC ??? 1-10a

■ **Performance of Parallel Algorithms**

* Assume that you are familiar with O, θ, Ω

$$\textbf{Speedup} = \frac{\text{worst-case running time of fastest known sequential algorithm}}{\text{worst-case running time of parallel algorithm}}$$

1-11a

*A parallel algorithm is said to achieve *linear speedup* if the speedup with p processor is $\theta(p)$.

Question: super linear speedup ?

Cost = (parallel running time)×(number of processors used) 1-11a

*A parallel algorithm is said to be *cost-optimal* if the cost matches the (sequential) lower bound to within a constant multiplicative factor.

fastest known
目前已知最快
(是否 optimal 會隨時間改變)

* Which one, speedup or cost, is more important ???

$$\textbf{\textit{Efficiency}} = \frac{\text{worst-case running time of fastest known sequential algorithm}}{\text{cost of parallel algorithm}}$$ 1-11a

* used only for experiments

= (speedup) / (number of used processors)

≤ 1 (100%, used for comparing the real running time)

$\theta(1)$, an optimal algo has an efficiency of $\theta(1)$

Example: Consider the odd-even transposition sort on a linear array of n processors, which sort n data in $O(n)$ time. 1-11c

Speedup = $(n\log n)/n = \log n$ (all in θ-notation)
cost = $n\times n = n^2$ (non-optimal!)
efficiency = $\log n/n$

* Note: treat all O as θ

Another example:
sort n numbers in $O(n^{0.5})$ time using $O(n^{0.5}\lg n)$ PEs

Question 1: cost optimal --> linear speedup ?

Question 2: cost optimal <-- linear speedup ?

cost optimal <--> linear speedup <--> efficiency = $\theta(1)$