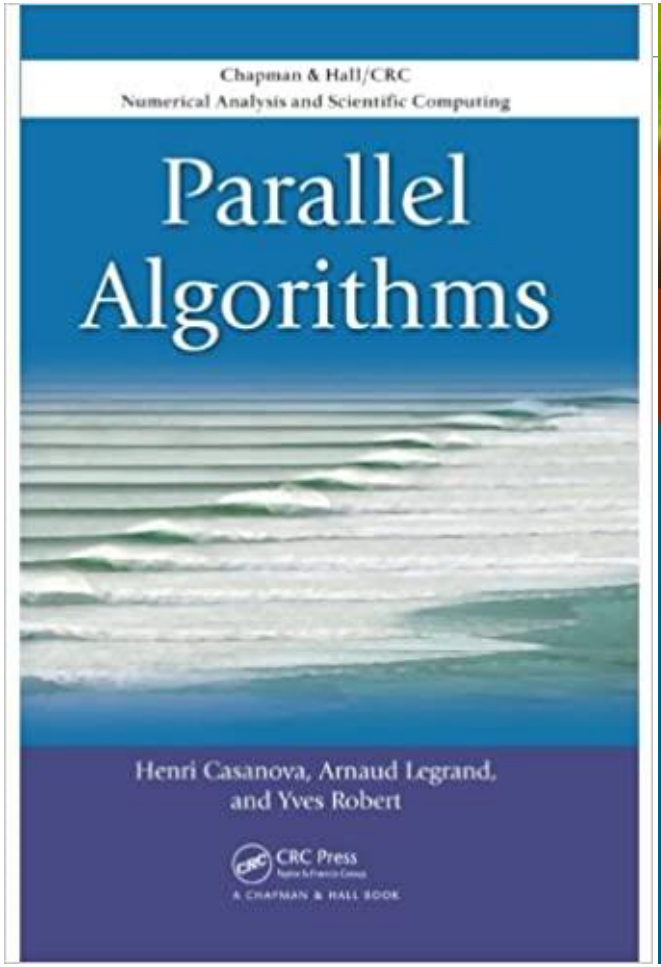
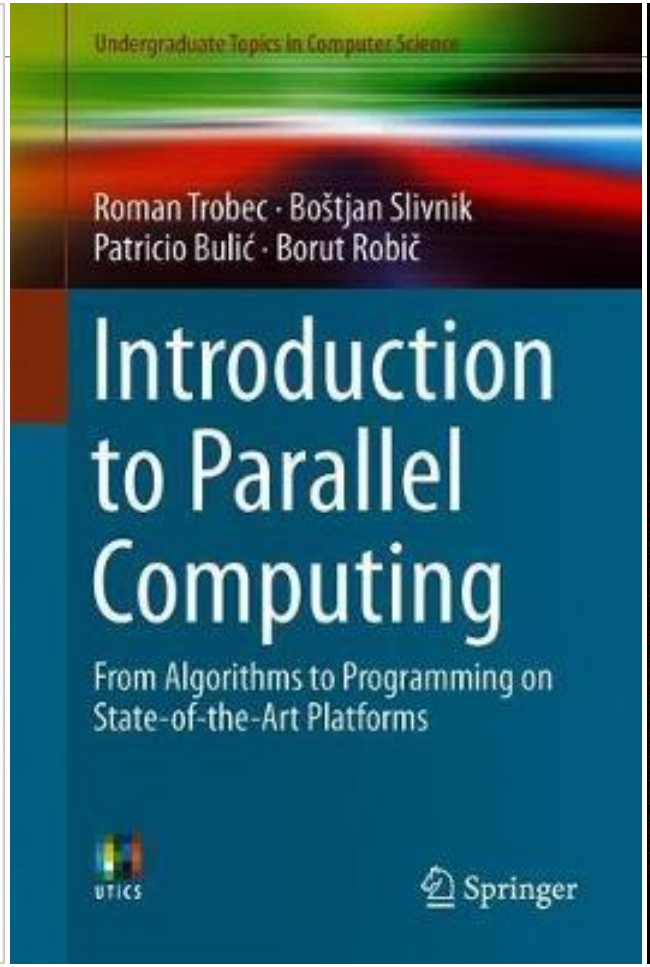
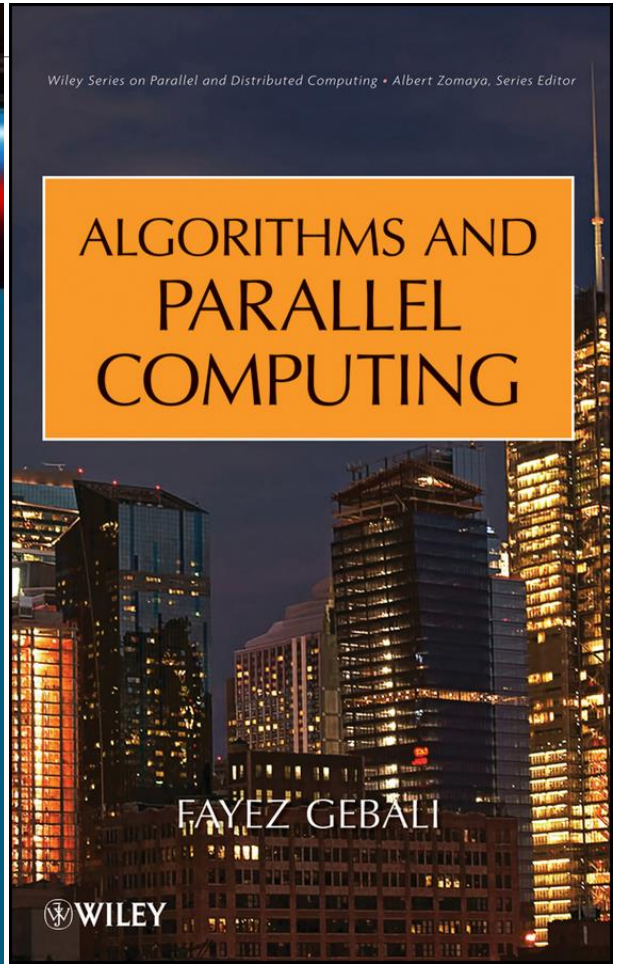
Parallel Programming

Lec 1

Books

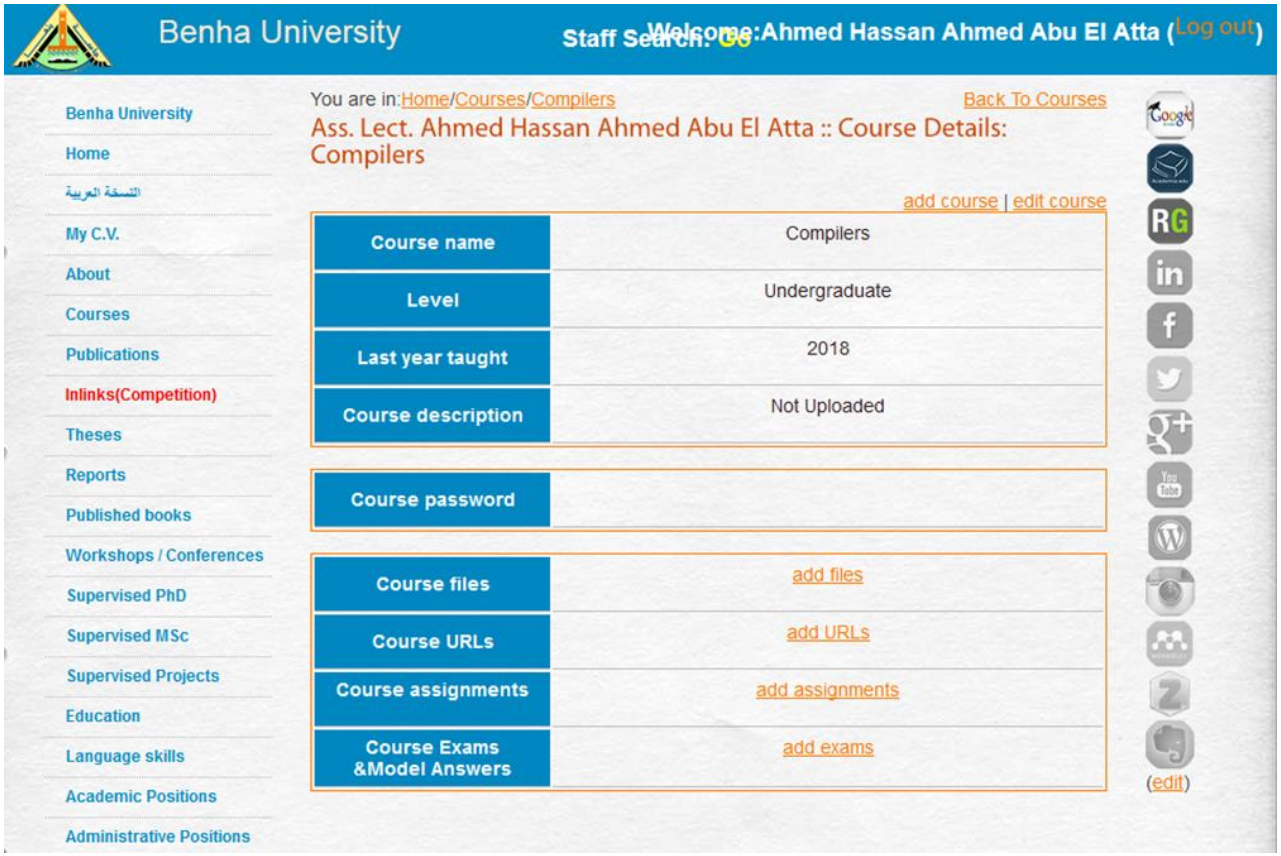

Chapman & Hall/CRC
Numerical Analysis and Scientific Computing
Parallel Algorithms
Henri Casanova, Arnaud Legrand, and Yves Robert
CRC Press
A CHAPMAN & HALL BOOK


Undergraduate Topics in Computer Science
Roman Trobec · Boštjan Slivnik
Patricio Bulić · Borut Robič
Introduction to Parallel Computing
From Algorithms to Programming on State-of-the-Art Platforms
UTICS
Springer


Wiley Series on Parallel and Distributed Computing • Albert Zomaya, Series Editor
ALGORITHMS AND PARALLEL COMPUTING
FAYEZ GEBALI
WILEY

PowerPoint

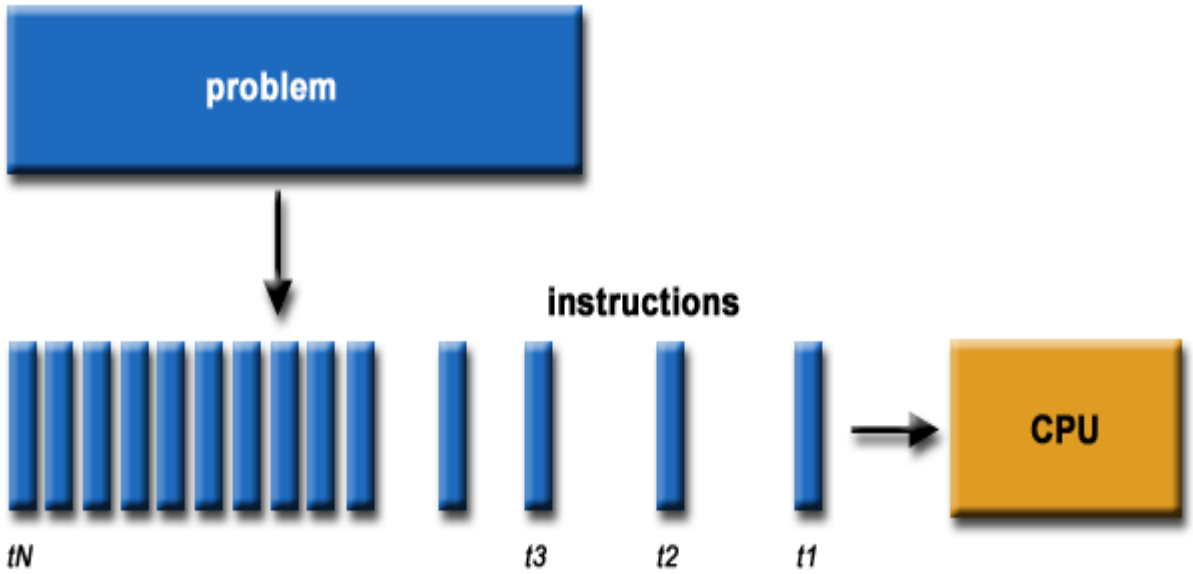
http://www.bu.edu.eg/staff/ahmedaboalatah14-courses/14779



Sequential Computation

Traditionally, software has been written for serial computation:

- To be run on a single computer having a single Central Processing Unit (CPU);
- A problem is broken into a discrete series of instructions.
- Instructions are executed one after another.
- Only one instruction may execute at any moment in time.

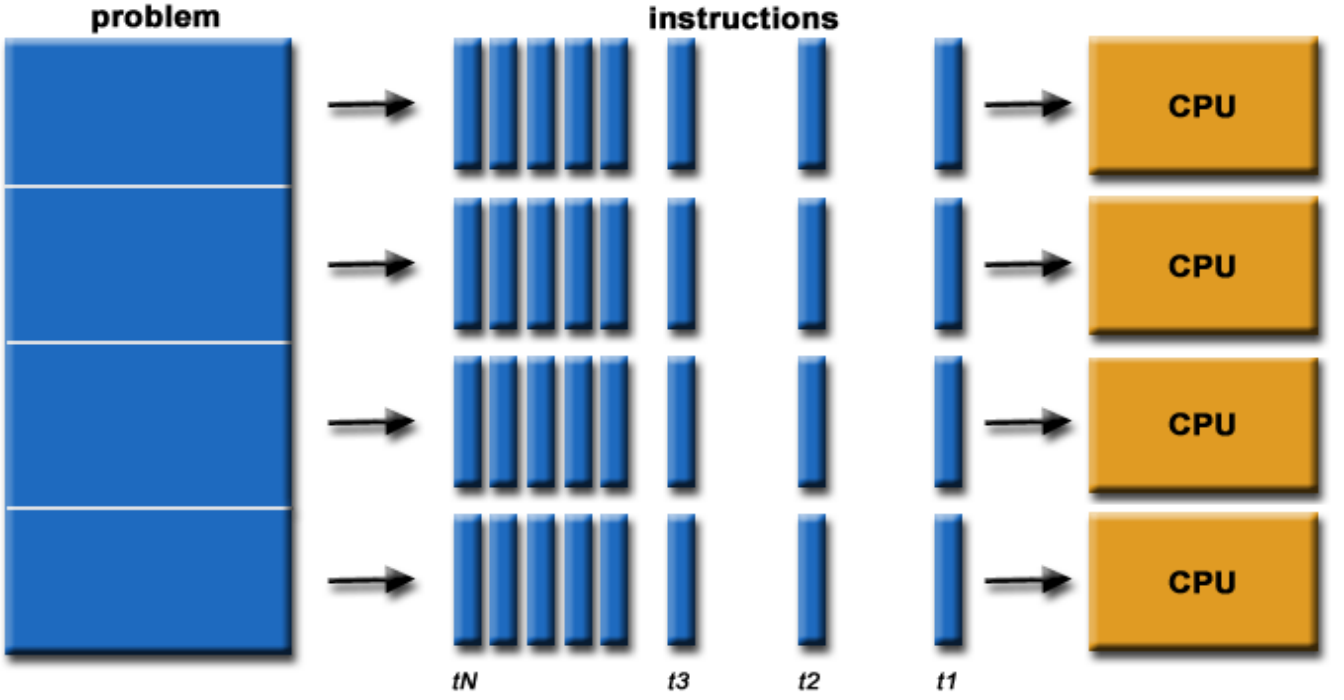


Parallel Computing

A parallel computer is a computer consisting of two or more processors that can cooperate and communicate to solve large problem fast

One or more memory modules

An interconnection network that connects processors with each other and/or with the memory modules.



Memory Architectures

Shared memory

Distributed memory

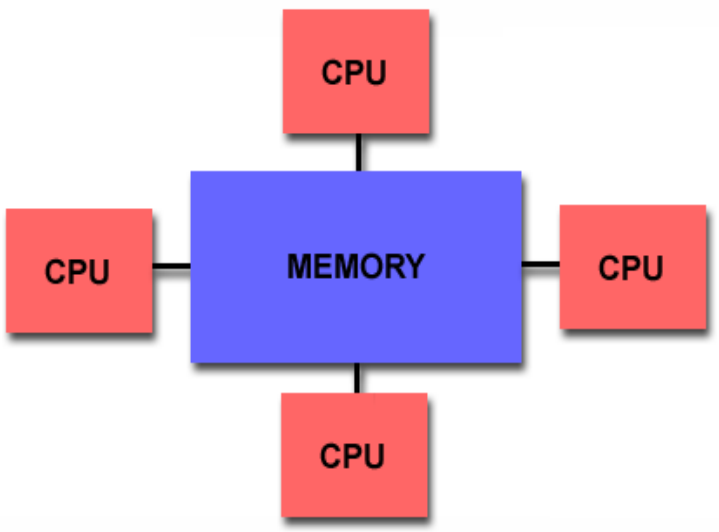
Hybrid distributed-shared memory solutions

Shared Memory

The Parallel Random Access Machine(PRAM)

Multiple processors can operate independently but share the same memory resources.

Changes in a memory location effected by one processor are visible to all other processors (global address space).



The variants of PRAM

1. Exclusive Read Exclusive Write PRAM (EREW-PRAM)
2. Concurrent Read Exclusive Write PRAM (CREW-PRAM)
3. Concurrent Read Concurrent Write PRAM (CRCW-PRAM)
4. Exclusive Read Concurrent Write PRAM (ERCW-PRAM)

Concurrent Read Concurrent Write PRAM (CRCW-PRAM)

CONSISTENT-CRCW-PRAM. Processing units may simultaneously attempt to write to L, *but it is assumed that they all need to write the same value to L*.

ARBITRARY-CRCW-PRAM. Processing units may simultaneously attempt to write to L (not necessarily the same value), but it is assumed that *only one of them will succeed*. *Which processing unit will succeed is not predictable*.

PRIORITY-CRCW-PRAM. There is a priority order imposed on the processing units; e.g., the processing unit with smaller index has higher priority. Processing units may simultaneously attempt to write to L, *but it is assumed that only the one with the highest priority will succeed*.

FUSION-CRCW-PRAM. Processing units may simultaneously attempt to write to L:

- the sum (+), product (*), maximum (max), minimum (min), logical conjunction (^), and logical disjunction ($\vee$).

Distributed Memory

Distributed memory systems require a communication network to connect inter - processor memory

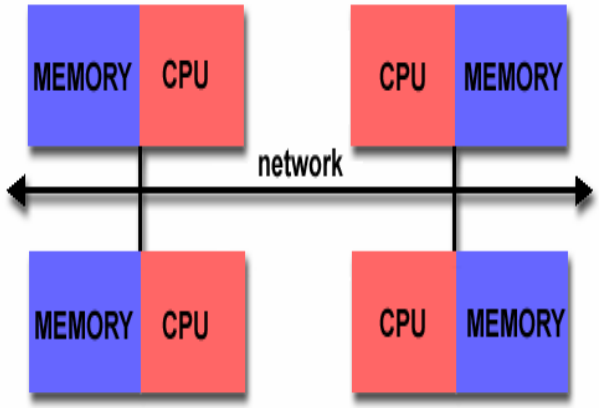
Processors have their own local memory, so

- it operates independently.
- Changes it makes to its local memory have no effect on the memory of other processors.

When a processor needs access to data in another processor, it is usually the task of the programmer to define how and when data is communicated.

Synchronization between tasks is

the programmer's responsibility.



Comparison

Shared Memory

Advantages

- Global address space
- Data sharing between tasks is both fast and uniform
  - due to the proximity of memory to CPUs

Disadvantages

- Lack of scalability between memory and CPUs.
  - Adding more CPUs can increases traffic on the shared memory and CPU path
- Programmer responsibility for synchronization
- Expensive

Distributed Memory

Advantages

- Memory is scalable with number of processor
- Each processor can rapidly access own memory

Disadvantages

- Programmer responsible for many details

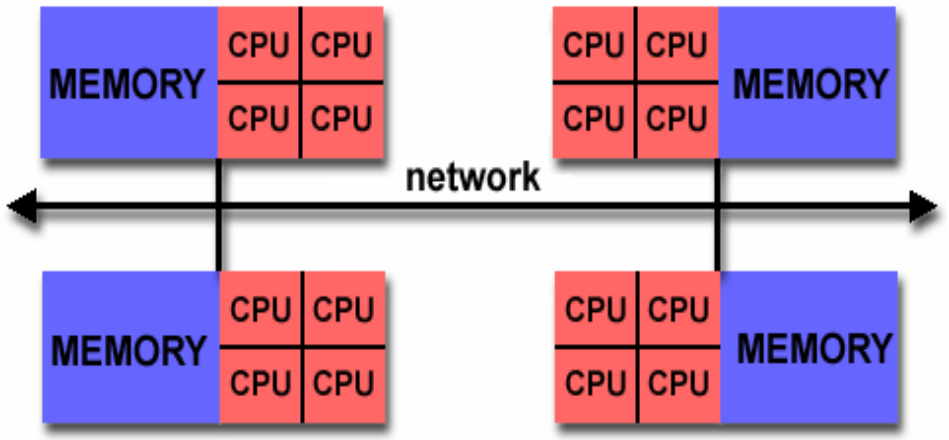
Hybrid Distributed-Shared Memory

Used in most of todays parallel computers.

The shared memory component is usually a cache coherent SMP machine.

The distributed memory component is the networking of multiple SMPs. SMPs know only about their own memory - not the memory on another SMP.

network communications are required to move data from one SMP to another.



Computer Architecture

- **Multiple processors:**
  - Flynn's taxonomy

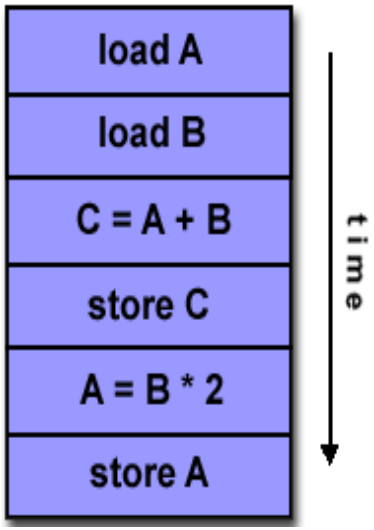
SISD Single Instruction, Single Data	**SIMD** Single Instruction, Multiple Data
MISD Multiple Instruction, Single Data	**MIMD** Multiple Instruction, Multiple Data

Michael Flynn's Classification

(1)instruction streams (2) data streams

SISD single instruction & single data

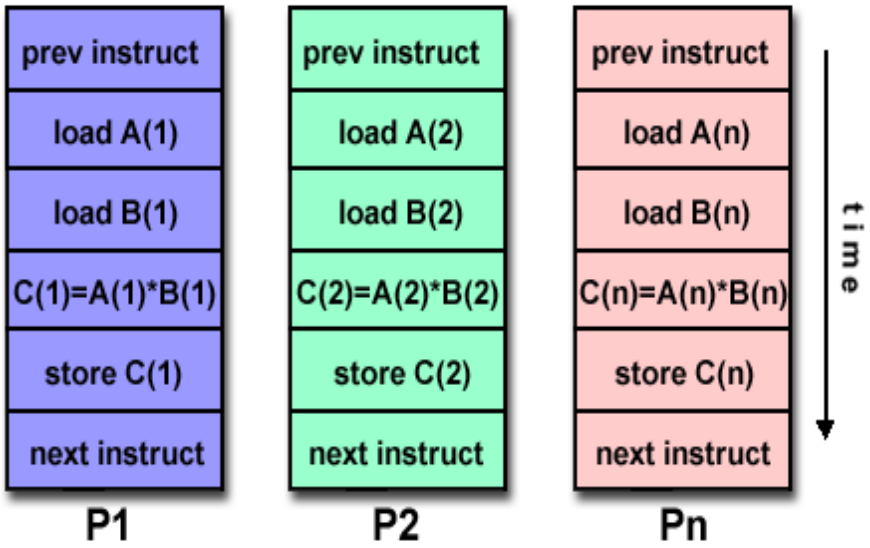
- A serial (non-parallel) computer.
- Single instruction: only one instruction stream is being acted on by the CPU during any one clock cycle.
- Single data: only one data stream is being used as input during any one clock cycle.
- This is the oldest and even today, the most common type of computer
- Examples: older generation mainframes, minicomputers and workstations; most modern day PCs.



Michael Flynn's Classification

SIMD single instruction & multiple data

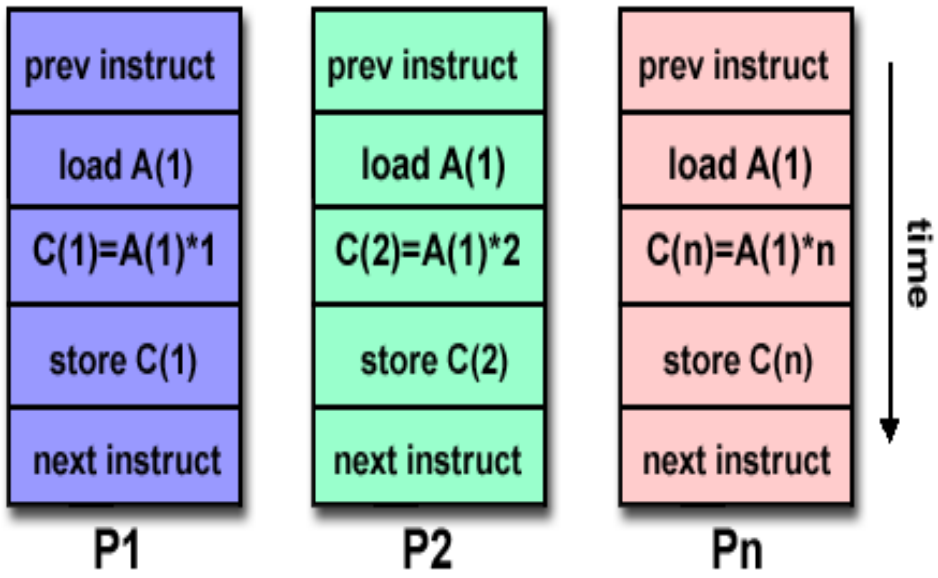
- A type of parallel computer.
- Single instruction: All processing units execute the same instruction at any given clock cycle
- Multiple data: Each processing unit can operate on a different data element
- Most modern computers, particularly those with graphics processor units (GPUs)



Michael Flynn's Classification

MISD multiple instruction & single data

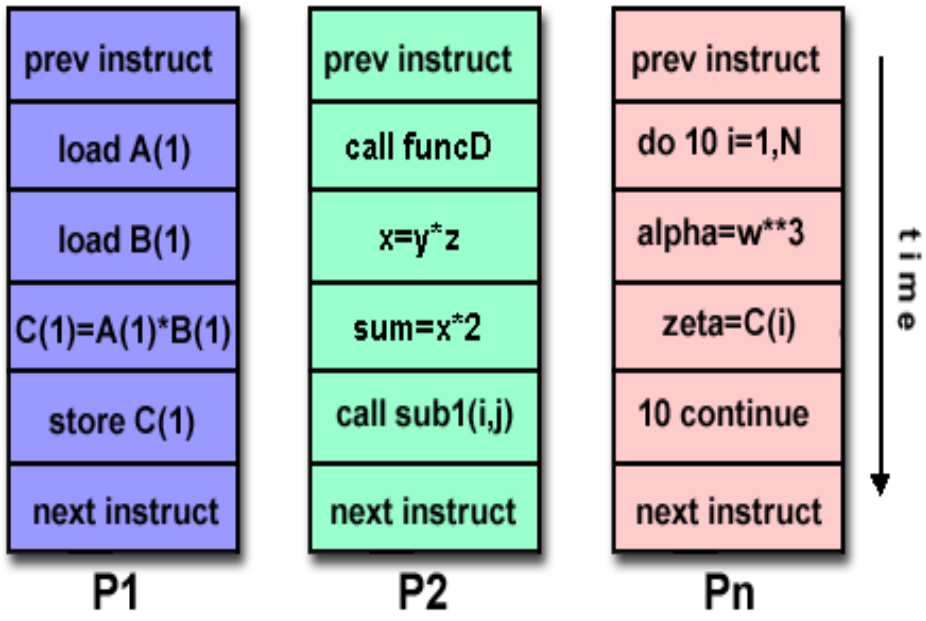
- A single data stream is fed into multiple processing units.
- Each processing unit operates on the data independently via independent instruction streams.



Michael Flynn's Classification

MIMD multiple instruction & multiple data

- The most common type of parallel computer.
- Multiple Instruction: every processor may be executing a different instruction stream
- Multiple Data: every processor may be working with a different data stream
- Examples: most current supercomputers, networked parallel computer "grids" and multi-processor SMP computers - including some types of PCs.



Why Use Parallel Computing?

- Save time
  - Solve a problem faster
- Solve larger problems (more memory)
- Provide concurrency
  - Doing many things simultaneously
- Economic limits
  - Fast single processor computers with large amounts of memory are expensive.

Performance Evaluation of PRAM Algorithms

Let ***P*** be a problem of size ***n*** that we want to solve (e.g., a computation over an n-element list).

Let $T_{seq}(n)$ be the execution time of the best (known) sequential algorithm for solving ***P***.

Let us now consider a PRAM algorithm that solves ***P*** in time $T_{par}(p, n)$ with ***p*** CPUs.

The cost of a PRAM algorithm is defined as $C_p(n) = p.T_{par}(p, n)$.

The work $W_p(n)$ of a PRAM algorithm is the sum over all CPUs of the number of performed operations. The difference between cost and work is that the work does not account for CPU idle time.

Performance Evaluation of PRAM Algorithms

The speedup of a PRAM algorithm is defined as

$$S_p(n) = T_{seq}(n) / T_{par}(p, n) \ .$$

The efficiency of the PRAM algorithm is defined as

$$E_p(n) = S_p(n) / p = T_{seq}(n) / p.T_{par}(p, n) = T_{seq}(n) / C_p(n).$$

Since $T_{par} \leq T_{seq} \leq p.T_{par}$, it follows that speedup is bounded above by and efficiency is bounded above by $E \leq 1$

Compute the summation of an array of integer numbers

Suppose that we are given the problem P ≡ “add n given numbers.”

Then “add numbers 1, 2, 3, 4, 5, 6, 7, 8” is an instance of size = n = 8 of the problem P.

Let us now focus on all instances of size 8, that is, instances of the form “add numbers $a_1;a_2;a_3;a_4;a_5;a_6;a_7;a_8$.”

The fastest sequential algorithm for computing the sum

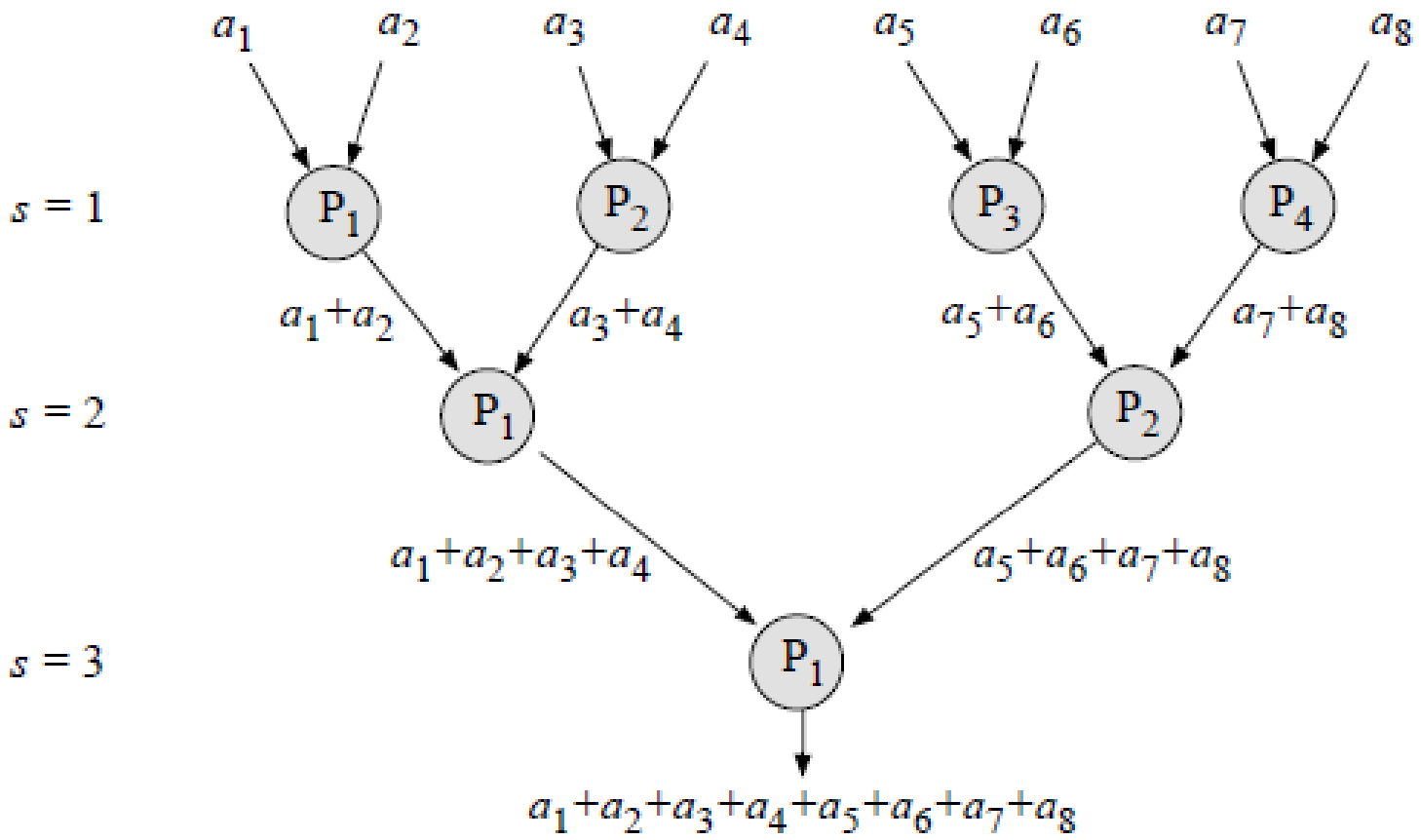
Sum = 0

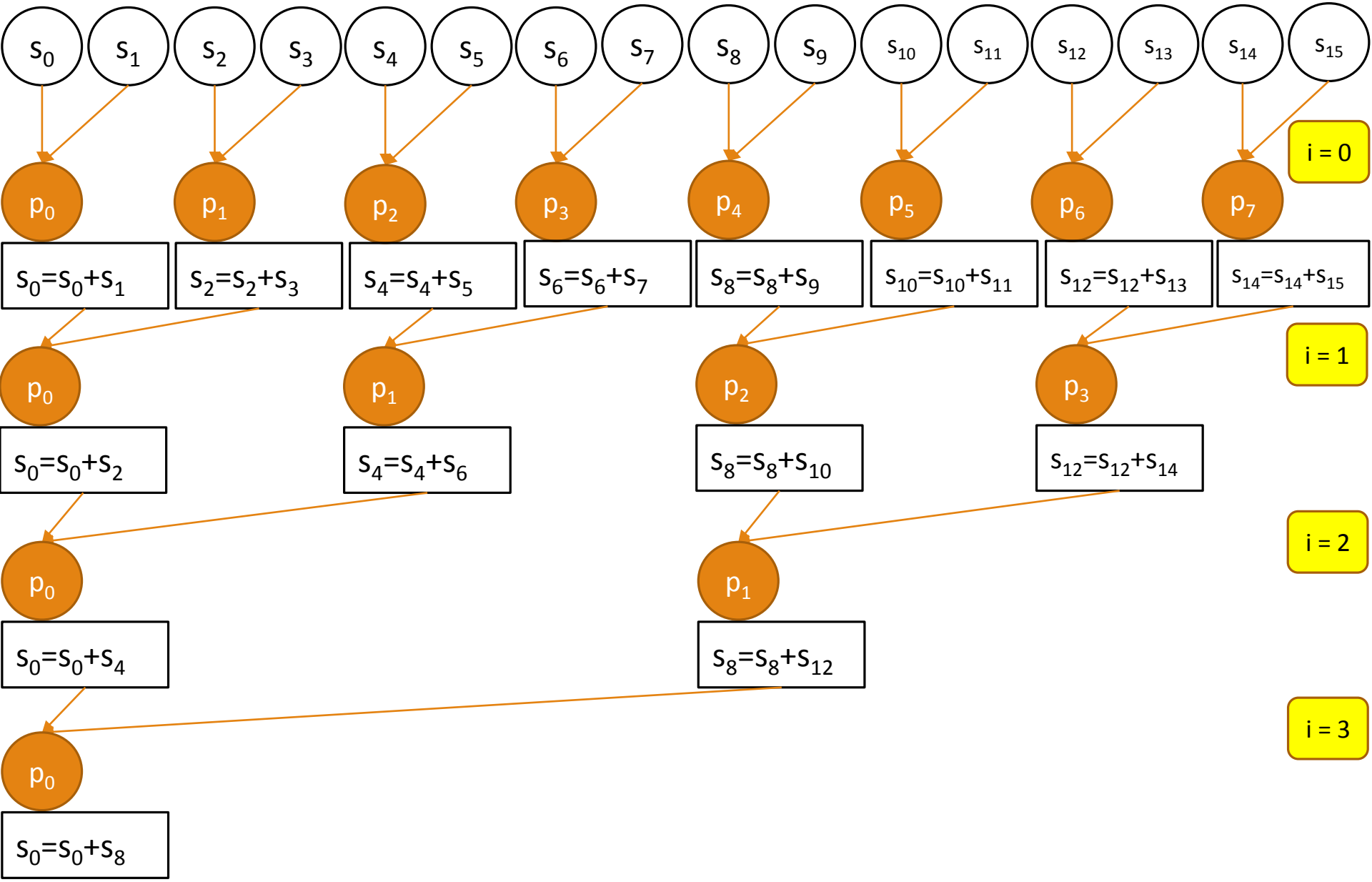
for (i = 1; i ≤ 8; i++)

sum += a_i

requires $T_{seq}(8) = 7$ steps → $T_{seq}(n) = O(n)$

Compute the summation of an array of integer numbers




s_0 s_1 s_2 s_3 s_4 s_5 s_6 s_7 s_8 s_9 s_{10} s_{11} s_{12} s_{13} s_{14} s_{15}
i = 0
p_0 p_1 p_2 p_3 p_4 p_5 p_6 p_7
$s_0=s_0+s_1$ $s_2=s_2+s_3$ $s_4=s_4+s_5$ $s_6=s_6+s_7$ $s_8=s_8+s_9$ $s_{10}=s_{10}+s_{11}$ $s_{12}=s_{12}+s_{13}$ $s_{14}=s_{14}+s_{15}$
i = 1
p_0 p_1 p_2 p_3
$s_0=s_0+s_2$ $s_4=s_4+s_6$ $s_8=s_8+s_{10}$ $s_{12}=s_{12}+s_{14}$
i = 2
p_0 p_1
$s_0=s_0+s_4$ $s_8=s_8+s_{12}$
i = 3
p_0
$s_0=s_0+s_8$

Compute the summation of an array of integer numbers

Sum = 0

For j = 0 to j < n/2 do parallel

    For i = 0 to i < 2 do

        s[j*2+i] = a[j*2+i]

For i = 0 to i<log(n) do

    For j = 0 to j <n/($2^{(i+1)}$) do in parallel

        s[j * $2^{(i+1)}$] += s[j * $2^{(i+1)}$+2^{i}]

sum = s[0]

Compute the summation of an array of integer numbers

In general, instances of size n of P can be solved in parallel time $T_{par} = O(\log n)$

speedup is $S(n) = T_{seq}(n) / T_{par}(n) = O(n/\log n)$.

Cost $C(n) = n*O(\log n) = O(n\log n)$

$E(n) = T_{seq}(n) / C(n) = O(n / (n\log n)) = O(1/\log n) < 1$