

Parallel and Distributed Computing

CS3006

Lecture 1

Introduction

14th February 2022

Dr. Rana Asif Rehman

CS30
06 -
Spring
2022

Aim of the course

to understand the **fundamental concepts** of parallel and distributed computing

design and analysis of **Parallel algorithms**

analyze different problems and **develop parallel programming solutions** of those problems

Study the **challenges of Parallel and Distributed systems** and how to cope with them



Outline

- ? **Motivating Parallelism**
- ? **Computing vs Systems**
- ? **Parallel vs Distributed Computing**
- ? **Practical Applications of P&D Computing**

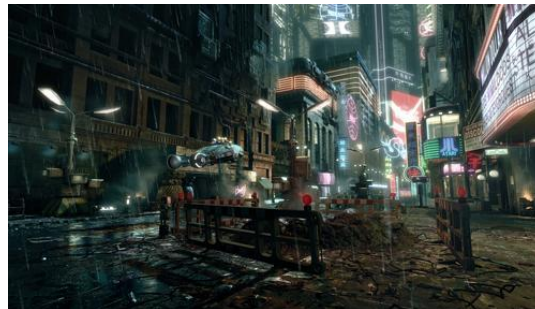
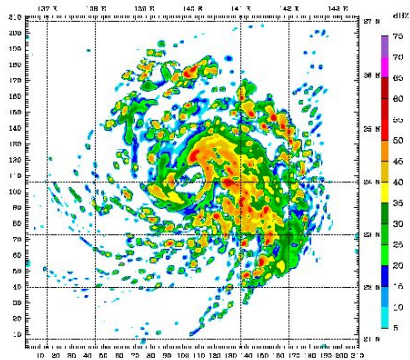


Motivating Parallelism

CS3006 - Spring 2022

Motivation for Parallel and Distributed Computing

- ? Uniprocessor are fast but
 - ? Some problems require too much computation
 - ? Some problems use too much data
 - ? Some problems have too many parameters to explore
- ? For example
 - ? Weather simulations, gaming, web Servers, code breaking



Motivating Parallelism

- ? Developing parallel hardware and software has traditionally been time and effort intensive.
- ? If one is to view this in the context of rapidly improving uniprocessor speeds, one is tempted to question the need for parallel computing.
- ? Latest trends in hardware design indicate that uni-processors may not be able to sustain the rate of *realizable* performance increments in the future .
- ? This is the result of a number of fundamental physical and computational limitations.
- ? The emergence of standardized parallel programming environments, libraries, and hardware have significantly reduced time to develop (parallel) solution.

Motivating Parallelism



Moore's Law

? Proposed by Gordon E. Moore in 1965 and revised in 1975.

? It states that **[Simplified Version]**

“Processing speeds, or overall processing power for computers will double every 18 months.”

? **A more technically correct interpretation**

“The number of transistors on an affordable CPU would double every two years [18 months].”

? Number of transistors incorporated in a chip will approximately double every two years

Our World
in Data

Motivating Parallelism



Moore's Law

- ? More computational power implicitly means more transistors.
- ? Then why need second interpretation?
- ? Let's have a look on empirical data from 1970 to 2009
 - ? In 1970's (i.e., from 1970 to 1979), processor speeds ranged from 740 KHz to 8 Mhz. Difference shows that both the interpretations are correct.
 - ? From 2000 to 2009, Speeds ranged from 1.3 GHz to 2.8 GHz.
 - ? Speed difference is too low but, number of integrated transistors ranged from 37.5 million to 904 million.
 - ? So, second interpretation is more accurate.

Motivating Parallelism

Moore's Law

- ? Why doubling the transistors does not double the speed?
 - ? The answer is increase in number of transistor per processor is due to multi-core CPU's.
 - ? It means, to follow Moore's law, companies had to:
 - ? Introduce ULSI(ultra large-scale integrations)
 - ? And multi-core processing era.
- ? Will Moore's law hold forever?
 - ? Adding multiple cores on single chip causes heat issues.
 - ? Furthermore, increasing the number of cores, may not be able to increase speeds [Due to inter-process interactions].
 - ? Moreover, transistors would eventually reach the limits of miniaturization at atomic levels

Motivating Parallelism

Moore's Law

- ? So, we must look for efficient parallel software solutions to fulfill our future computational needs.
- ? As stated earlier, number of cores on a single chip also have some restrictions.
- ? Solution[s]?
 - ? Need to find more scalable distributed and hybrid solutions

Motivating Parallelism



The Memory/Disk Speed Argument

- ? While clock rates of high-end processors have increased at roughly 40% per year over the past decade, DRAM access times have only improved at the rate of roughly 10% per year over this interval.
- ? This mismatch in speeds causes significant performance bottlenecks.
- ? Parallel platforms provide increased bandwidth to the memory system.
- ? Parallel platforms also provide higher aggregate caches.
- ? Some of the fastest growing applications of parallel computing utilize not their raw computational speed, rather their ability to pump data to memory and disk faster.

Motivating Parallelism



The Data Communication Argument

- ? As the network evolves, the vision of the Internet as one large computing platform has emerged.
- ? In many applications like databases and data mining problems, the volume of data is such that they cannot be moved.
- ? Any analyses on this data must be performed over the network using parallel techniques

Computing vs Systems

Distributed Systems

- ? A collection of autonomous computers, connected through a network and distribution middleware.
 - ? This enables computers to coordinate their activities and to share the resources of the system.
 - ? The system is usually perceived as a single, integrated computing facility.
 - ? Mostly concerned with the hardware-based accelerations

Distributed Computing

- ? A specific use of distributed systems, to split a large and complex processing into subparts and execute them in parallel, to increase the productivity.
 - ? Computing mainly concerned with software-based accelerations (i.e., designing and implementing algorithms)

Parallel vs Distributed Computing

Parallel (shared-memory) Computing

- ? The term is usually used for developing concurrent solutions for following two types of the systems:
 1. Multi-core Architecture
 2. Many core architectures (i.e., GPU's)

Distributed Computing

- ? This type of computing is mainly concerned with developing algorithms for the distributed cluster systems.
- ? Here distributed means a geographical distance between the computers without any shared-Memory.

Practical Applications of P&D Computing



Scientific Applications

- ? Functional and structural characterization of genes and proteins
- ? Applications in astrophysics have explored the evolution of galaxies, thermonuclear processes, and the analysis of extremely large datasets from telescope.
- ? Advances in computational physics and chemistry have explored new materials, understanding of chemical pathways, and more efficient processes
 - ? e.g., Large Hydron Collider (LHC) at European Organization for Nuclear Research (CERN) generates petabytes of data for a single collision.

Practical Applications of P&D Computing

Scientific Applications

- ? Bioinformatics and astrophysics also present some of the most challenging problems with respect to analyzing extremely large datasets.
- ? Weather modeling for simulating the track of a natural hazards like the extreme cyclones (storms).
- ? Flood prediction

Practical Applications of P&D Computing

Commercial Applications

- ? Some of the largest parallel computers power the wall street!
- ? Data mining-analysis for optimizing business and marketing decisions.
- ? Large scale servers (mail and web servers) are often implemented using parallel platforms.
- ? Applications such as information retrieval and search are typically powered by large clusters.

Practical Applications of P&D Computing

Computer Systems Applications

- ? Network intrusion detection: A large amount of data needs to be analyzed and processed
- ? Cryptography (the art of writing or solving codes) employs parallel infrastructures and algorithms to solve complex codes.
- ? Graphics processing
- ? Embedded systems increasingly rely on distributed control algorithms. E.g. modern automobiles

Limitations of Parallel Computing:



- ? It requires designing the proper communication and synchronization mechanisms between the processes and sub-tasks.
- ? Exploring the proper parallelism from a problem is a hectic process.
- ? The program must have low coupling and high cohesion. But it's difficult to create such programs.
- ? It needs relatively more technical skills to code a parallel program.

Questions





Parallel and Distributed Computing

CS3006

Lecture 2

Amdahl's Law

21th February 2022

Dr. Rana Asif Rehman

Outline

- ? Amdahl's Law of Parallel Speedup
- ? Karp-Flatt Metric
- ? Types of Parallelism
 - ? Data-parallelism
 - ? Functional-parallelism
 - ? Pipelining
- ? Multi-processor vs Multi-computer
- ? Cluster vs Network of workstations

CS3006 - Fall 2021

Amdahl's Law

- Amdahl's was formulated in 1967
- It shows an upper-bound on the maximum speedup that can be achieved
- Suppose you are going to design a parallel algorithm for a problem
- Further suppose that ***fraction*** of total time that the algorithm must consume in **serial executions** is '**F**'
- This implies ***fraction*** of parallel portion is (1 - F)
- Now, Amdahl's law states that

$$\text{Speedup}(p) = \frac{1}{F + \frac{1-F}{P}}$$

- Here 'p' is total number of available processing nodes.

Amdahl's Law

Derivation

- Let's suppose you have a sequential code for a problem that can be executed in total **$T(s)$** time.
- $T(p)$** be the parallel time for the same algorithm over p processors.

Then speedup can be calculated using:-

$$\text{Speedup}(p) = \frac{T(s)}{T(p)}$$

- $T(p)$ can be calculated as:

$T(p) = \text{serial comput. time} + \text{Parallel comp. time}$

$$T(p) = F \cdot T(s) + \frac{(1-F) \cdot T(s)}{p}$$

Amdahl's Law

Derivation

➡ Again

$$Speedup(p) = \frac{T(s)}{T(p)} \Rightarrow \frac{T(s)}{F \cdot T(s) + \frac{(1-F) \cdot T(s)}{P}}$$

$$\Rightarrow Speedup(p) = \frac{1}{F + \frac{1-F}{P}}$$

➡ What if you have infinite number of processors?

Amdahl's Law

- **Example 1:** Suppose 70% of a sequential algorithm is parallelizable portion. The remaining part must be calculated sequentially. Calculate maximum theoretical speedup for parallel variant of this algorithm using i). 4 processors and ii). infinite processors.
- $F = 0.30$ and $1 - F = 0.70$ use Amdahl's law to calculate theoretical speedups.

Amdahl's Law

- ? **Example 2:** Suppose 25% of a sequential algorithm is parallelizable portion. The remaining part must be calculated sequentially. Calculate maximum theoretical speedup for parallel variant of this algorithm using 5 processors and infinite processors.
- ? ???
- ? **Little challenge:** Determine, according to Amdahl's law, how many processors are needed to achieve maximum theoretical speedup while sequential portion remains the same?
- ? The answer may be surprising?
- ? That's why we say actual achievable speedup is always less-than or equal to theoretical speedups.



Karp-Flatt Metric

CS3006 - Fall 2021

Karp-Flatt Metric

➤ The metric is used to calculate serial fraction for a given parallel configuration.

- i.e., if a parallel program is exhibiting a speedup **S** while using **P** processing units then experimentally determined serial fraction **e** is given by :-

$$e = \frac{1/S - 1/p}{1 - 1/p}$$

➤ **Example task:** Suppose in a parallel program, for 5 processors, you gained a speedup of 1.25x, determine sequential fraction of your program.

Solution: Compute $e(n, p)$ corresponding to each data point:

p	2	3	4	5	6	7	8
$\Psi(n, p)$	1.82	2.50	3.08	3.57	4.00	4.38	4.71
$e(n, p)$	0.1	0.1	0.1	0.1	0.1	0.1	0.1

Since the experimentally determined serial fraction $e(n, p)$ is not increasing with p , the primary reason for the poor speedup is the 10% of the computation that is inherently sequential. Parallel overhead is not the reason for the poor speedup.

Benchmarking a parallel program on 1, 2, ..., 8 processors produces the following speedup results:

p	2	3	4	5	6	7	8
$\Psi(n, p)$	1.87	2.61	3.23	3.73	4.14	4.46	4.71

What is the primary reason for the parallel program achieving a speedup of 4.71 on 8 processors?

Solution:

p	2	3	4	5	6	7	8
$\Psi(n, p)$	1.87	2.61	3.23	3.73	4.14	4.46	4.71
e	0.07	0.075	0.08	0.085	0.09	0.095	0.1

Since the experimentally determined serial fraction e is steadily increasing with p , parallel overhead also contributes to the poor speedup.



Types of Parallelism

Types of Parallelism

1. Data-parallelism

? When there are independent tasks applying the same operation to different elements of a data set

? Example code

for i=0 to 99 do

a[i] = b[i] + c[i]

Endfor

CS3006 - Fall 2021

? Here same operation addition is being performed on first 100 of 'b' and 'c'

? All 100 iterations of the loop could be executed

Types of Parallelism

2. Functional-parallelism

- When there are independent tasks applying different operations to different data elements
- Example code
 - 1) $a=2$
 - 2) $b=3$
 - 3) $m = (a+b)/2$
 - 4) $s = (a^2 + b^2)/2$
 - 5) $v = s - m^2$
- Here third and fourth statements could be performed concurrently.

Types of Parallelism

3. Pipelining

- ? Usually used for the problems where single instance of the problem can not be parallelized
- ? The output of one stage is input of the other stage
- ? Dividing whole computation of each instance into multiple stages provided that there are multiple instances of the problem
- ? An effective method of attaining parallelism on the uniprocessor architectures
- ? Depends on pipelining abilities of the processor

Types of Parallelism

3. Pipelining

? Example:
Assembly line
analogy

Time	Engine	Doors	Wheels	Paint
5 min	Car 1			
10 min		Car 1		
15 min			Car 1	
20 min				Car 1
25 min	Car 2			
30 min		Car 2		
35 min			Car 2	
40 min				Car 2

Sequential Execution

Types of Parallelism

3. Pipelining

? Example:
Assembly line
analogy

Time	Engine	Doors	Wheels	Paint
5 min	Car 1			
10 min	Car 2	Car 1		
15 min	Car 3	Car 2	Car 1	
20 min	Car 4	Car 3	Car 2	Car 1
25 min		Car 4	Car 3	Car 2
30 min			Car 4	Car 3
35 min				Car 4

Pipelining

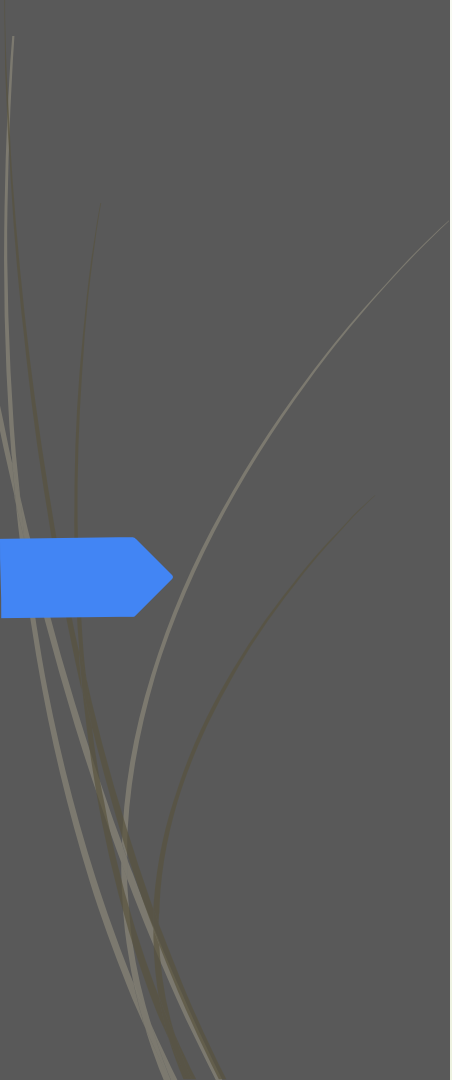
Types of Parallelism

3. Pipelining

? Example:
Overlap
instructions in a
single instruction
cycle to achieve
parallelism

Cycles	Fetch	Decode	Execute	Save
1	Inst 1			
2	Inst 2	Inst 1		
3	Inst 3	Inst 2	Inst 1	
4	Inst 4	Inst 3	Inst 2	Inst 1
5		Inst 4	Inst 3	Inst 2
6			Inst 4	Inst 3
7				Inst 4

4-stage Pipelining



Multi-processor vs Multi-Computer

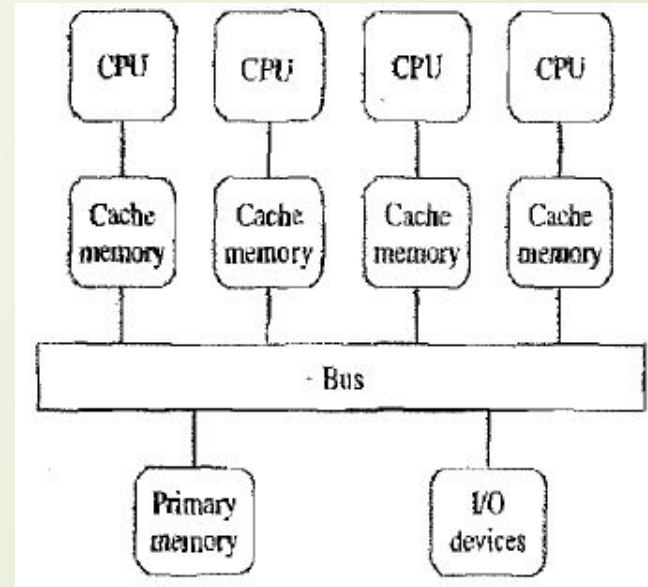
Multi-Processor

- ? Multiple-CPU's with a shared memory
- ? The same address on two different CPU's refers to the same memory location.
- ? **Generally two categories:-**
 1. Centralized Multi-processors
 2. Distributed Multi-processor

Multi-Processor

i. Centralized Multi-processor

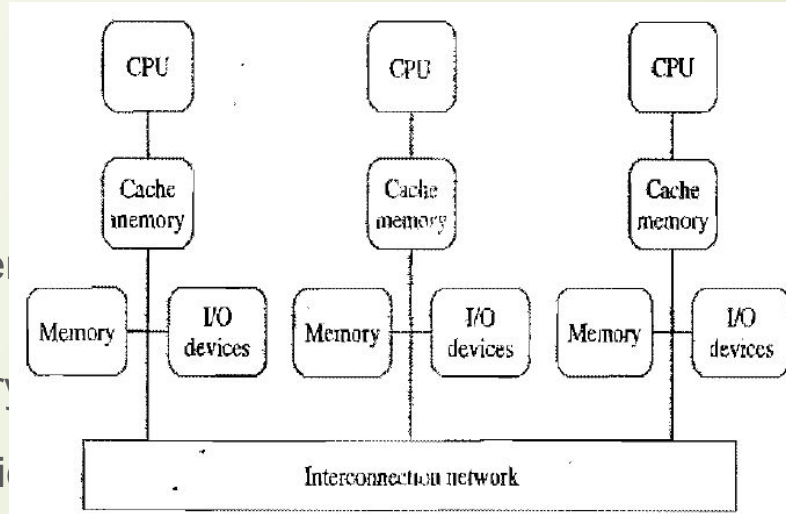
- ? Additional CPUs are attached to the system bus, and all the processors share the same primary memory
- ? All the memory is at one place and has the same access time from every processor
- ? Also known to as **UMA**(Uniform Memory Access) multi-processor or **SMP** (symmetrical Multi-processor)



Multi-Processor

ii. Distributed Multi-processor

- ? Distributed collection of memories forms one logical address space
- ? Again, the same address on different processors refers to the same memory location.
- ? Also known as non-uniform memory access (**NUMA**) architecture
- ? Because, memory access time varies significantly, depending on the physical location of the referenced address



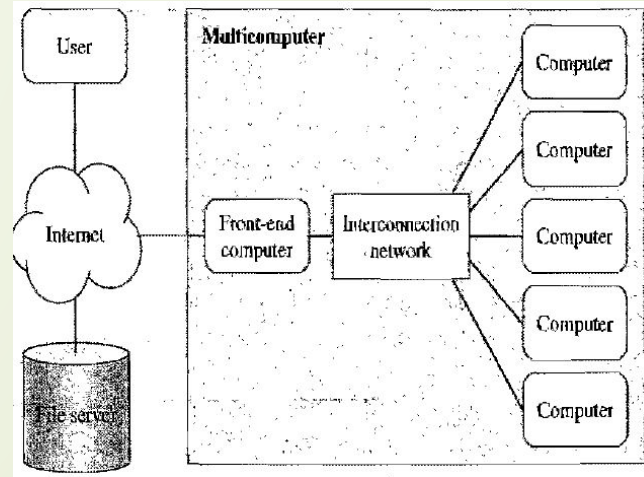
Multi-Computer

- ? Distributed-memory, multi-CPU computer
- ? Unlike **NUMA** architecture, a multicomputer has disjoint local address spaces
- ? Each processor has direct access to their local memory only.
- ? The same address on different processors refers to two different physical memory locations.
- ? Processors interact with each other through passing messages

Multi-Computer

Asymmetric Multi-Computers

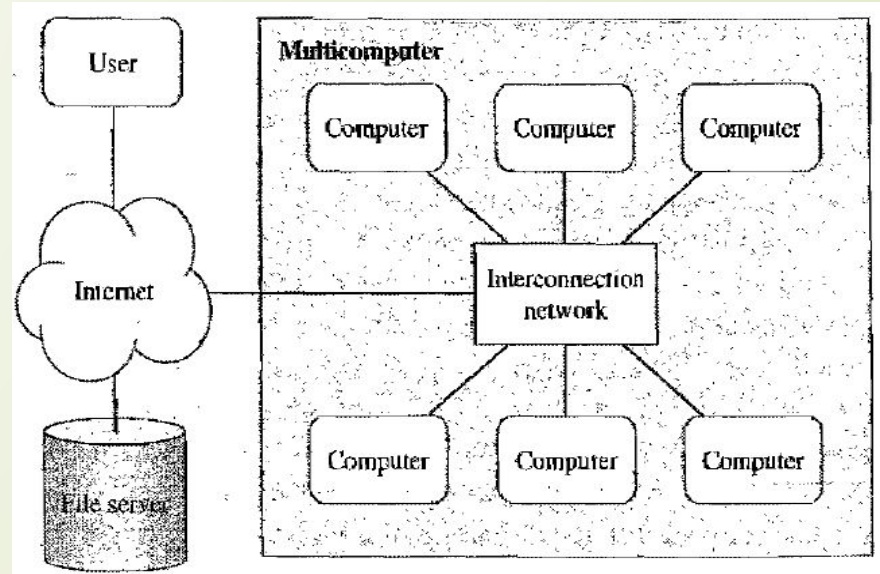
- ? A front-end computer that interacts with users and I/O devices
- ? The back-end processors are dedicatedly used for “number crunching”
- ? Front-end computer executes a full, multiprogrammed OS and provides all functions needed for program development
- ? The backends are reserved for executing parallel programs

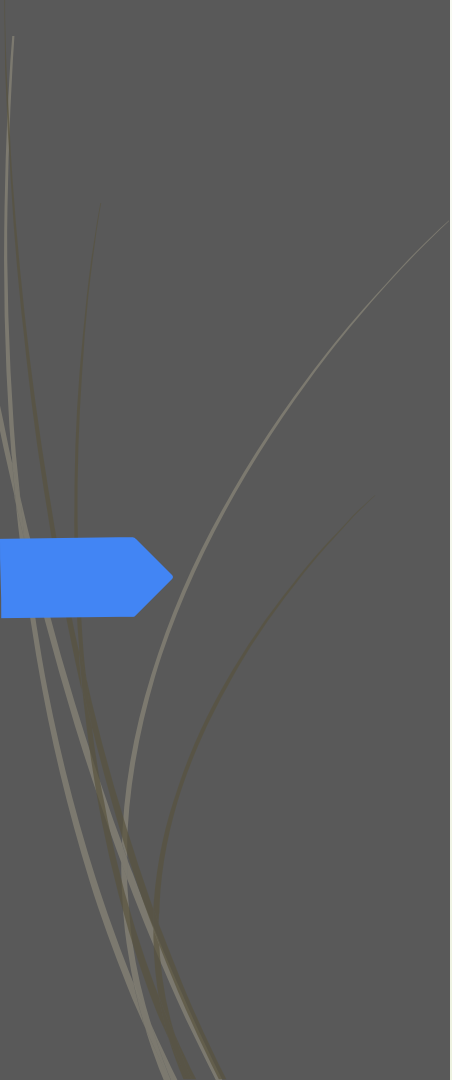


Multi-Computer

Symmetric Multi-Computers

- ? Every computer executes same OS
- ? Users may log into any of the computers
- ? This enables multiple users to concurrently login, edit and compile their programs.
- ? All the nodes can participate in execution of a parallel program





Network of Workstations vs Cluster

Cluster	Network of workstations
<p>Usually a co-located collection of low-cost computers and switches, dedicated to running parallel jobs. All computer run the same version of operating system.</p>	<p>A dispersed collection of computers. Individual workstations may have different Operating systems and executable programs</p>
<p>Some of the computers may not have interfaces for the users to login</p>	<p>User have the power to login and power off their workstations</p>
<p>Commodity cluster uses high speed networks for communication such as fast Ethernet@100Mbps, gigabit Ethernet@1000 Mbps and Myrinet@1920 Mbps.</p>	<p>Ethernet speed for this network is usually slower. Typical in range of 10 Mbps</p>



Reading Assignment

- ? Cache Coherence and Snooping
- ? Branch prediction and issues while
pipelining the problem

Assigned reading pointers:



? Cache Coherence:

- ? When we are in a distributed environment, each CPU's cache needs to be consistent (continuously needs to be updated for current values), which is known as cache coherence.

? Snooping:

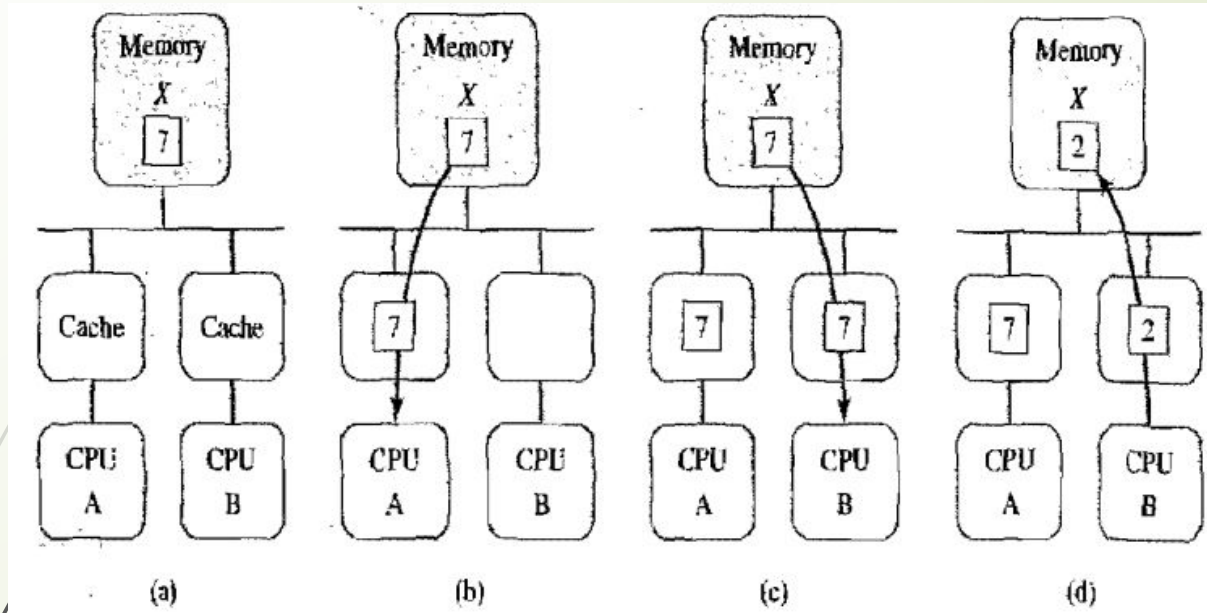
- ? Snoopy protocols achieve data consistency between the cache memory and the shared memory through a bus-based memory system. Write-invalidate and write-update policies are used for maintaining cache consistency.

? Branch Prediction:

- ? Branch prediction is a technique used in CPU design that attempts to guess the outcome of a conditional operation and prepare for the most likely result.

Questions







Parallel and Distributed Computing

CS3006

Lecture 3

Flynn's Taxonomy

23rd February 2022

Dr. Rana Asif Rehman

Agenda

- ? A Quick Review
- ? Flynn's Taxonomy
 - ? SISD
 - ? MISD
 - ? SIMD
 - ? MIMD
- ? Physical Organization of Parallel Platforms
 - ? PRAM
- ? Routing techniques and Costs

CS3006 - Spring 2022

Quick Review to the Previous Lecture

? **Amdahl's Law of Parallel Speedup**

- ? Purpose, derivation, and examples

? **Karp-Flatt Metric**

- ? Finding sequential fraction in the given parallel setup

? **Types of Parallelism**

- ? Data-parallelism

- ? Same operation on different data elements

- ? Functional-parallelism

CS3006 - Spring 2022
? Different independent tasks with different operations on different data elements can be parallelized

- ? Pipelining

- ? Overlapping the instructions in a single instruction cycle to achieve parallelism

Quick Review to the Previous Lecture

? **Multiprocessor**

- ? Centralized multiprocessor
- ? Distributed multiprocessor
- ? Shared address space(NUMA) vs Shared memory(UMA)

? **Multicomputer**

- ? Asymmetrical
- ? Symmetrical

? **Cluster vs Network of Workstations**

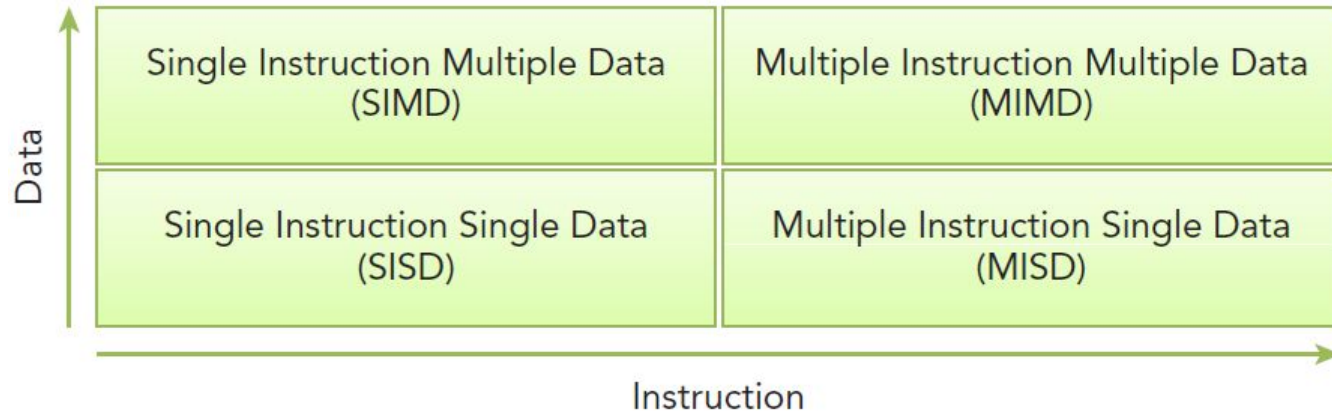
? **Assigned Reading**

- ? Cache Coherence and snooping
- ? Branch prediction and issues while pipelining the problem

CS3006 - Spring 2022

Flynn's Taxonomy

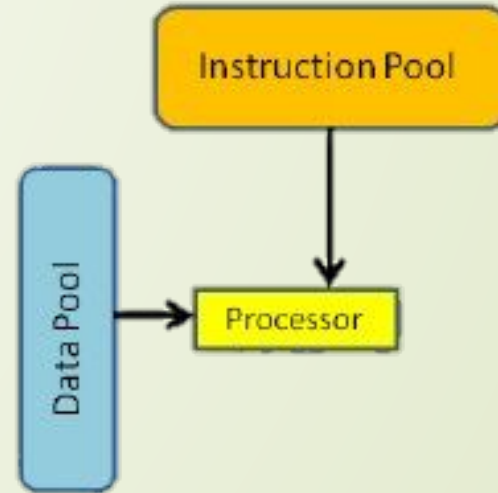
- ? Widely used architectural classification scheme
- ? Classifies architectures into four types
- ? The classification is based on how data and instructions flow through the cores.



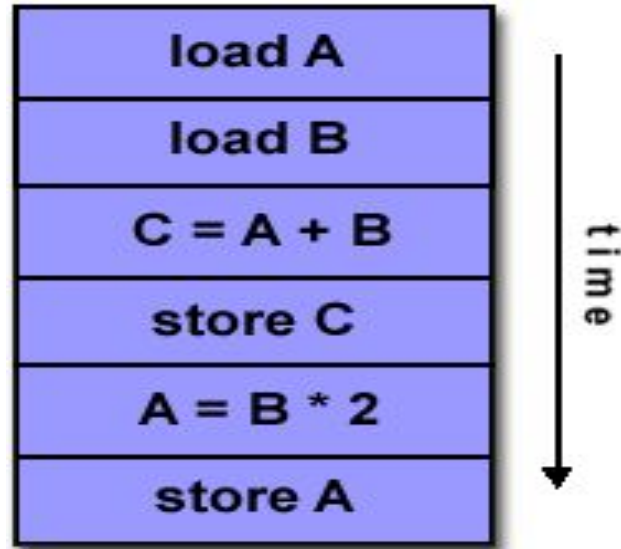
Flynn's Taxonomy

SISD (Single Instruction Single Data)

- ? Refers to traditional computer: a serial architecture
- ? This architecture includes single core computers
- ? Single instruction stream is in execution at a given time
- ? Similarly, only one data stream is active at any time



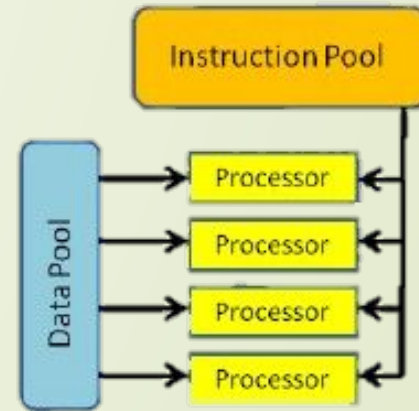
Example of SISD:



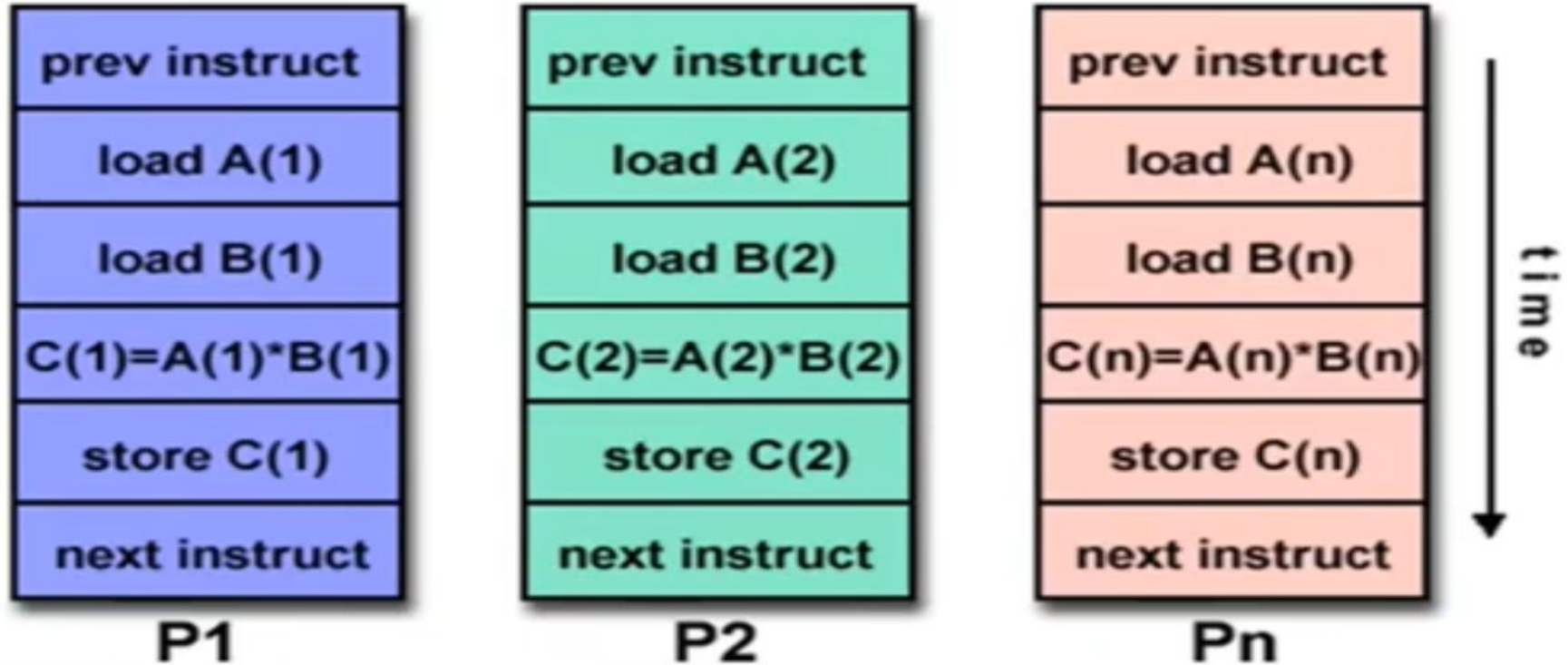
Flynn's Taxonomy

SIMD (Single Instruction Multiple Data)

- ? Refers to parallel architecture with multiple cores
- ? All the cores execute the same instruction stream at any time but, data stream is different for the each.
- ? Well-suited for the scientific operations requiring large matrix-vector operations
- ? Vector computers (Cray vector processing machine) and Intel co-processing unit 'MMX' fall under this category.
- ? Used with array operations, image processing and graphics



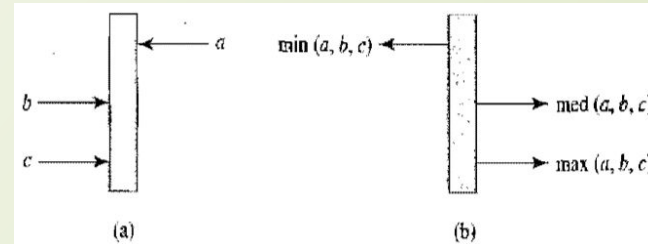
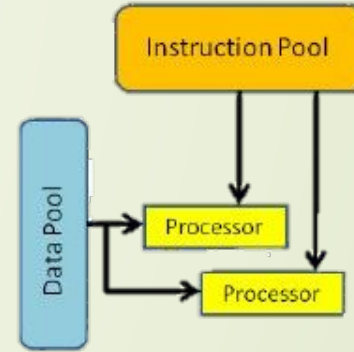
Example of SIMD:



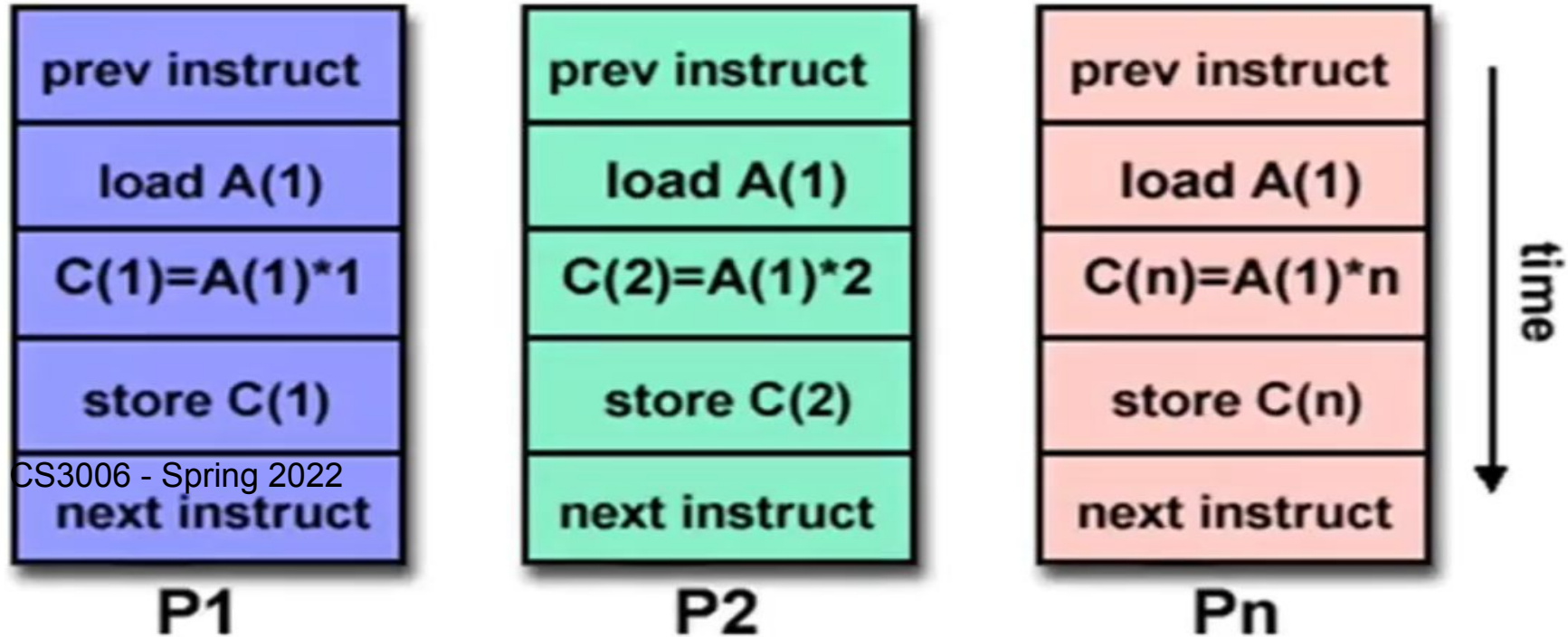
Flynn's Taxonomy

MISD (Multiple Instructions Single Data)

- ? Multiple instruction stream and single data stream
 - ? A pipeline of multiple independently executing functional units
 - ? Each operating on a single stream of data and forwarding results from one to the next
- ? Rarely used in practice
- ? E.g., Systolic arrays : network of primitive processing elements that pump data.



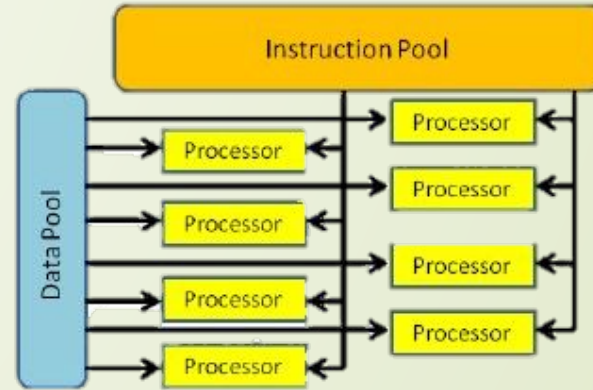
Example of MISD:



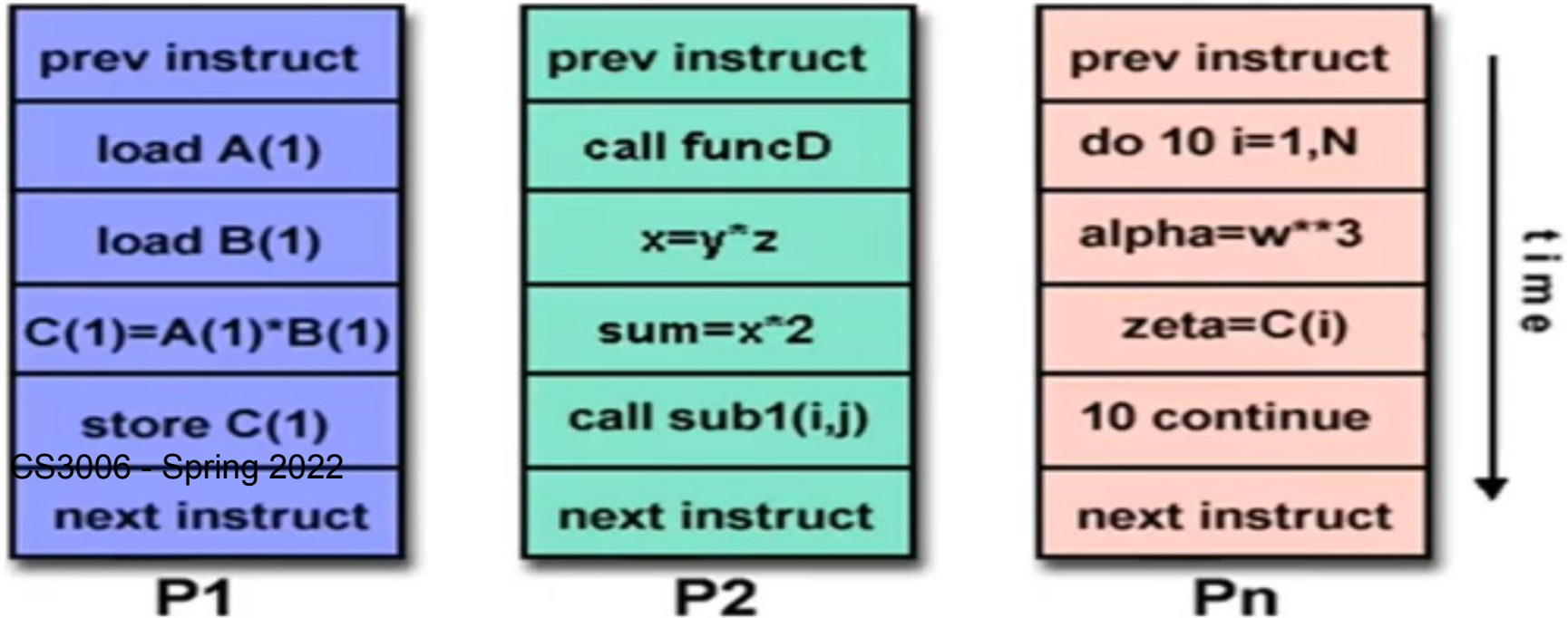
Flynn's Taxonomy

MIMD (Multiple Instructions Multiple Data)

- ? Multiple instruction streams and multiple data streams
- ? Different CPUs can simultaneously execute different instruction streams manipulating different data
- ? Most of the modern parallel architectures fall under this category e.g., **Multiprocessor** and **multicomputer** architectures
- ? Many MIMD architectures include SIMD executions by default.

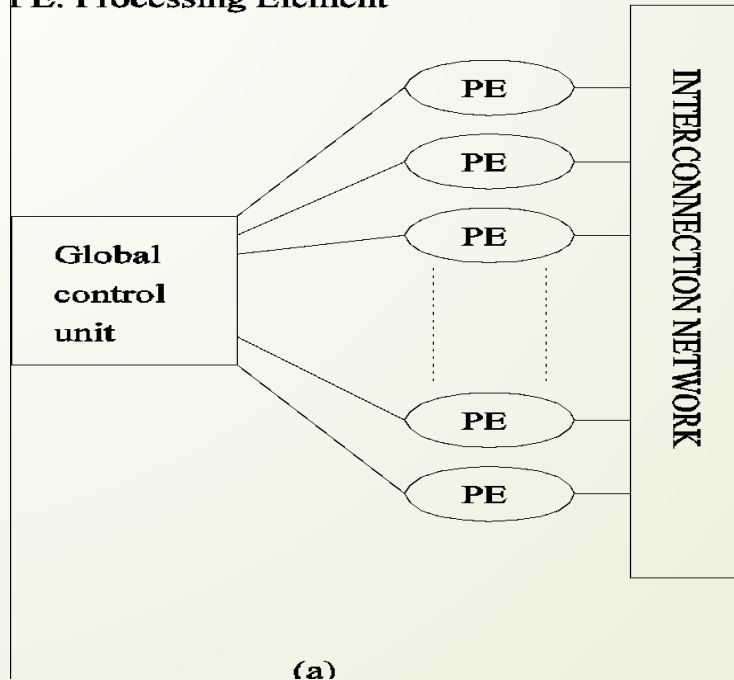


Example of MIMD:

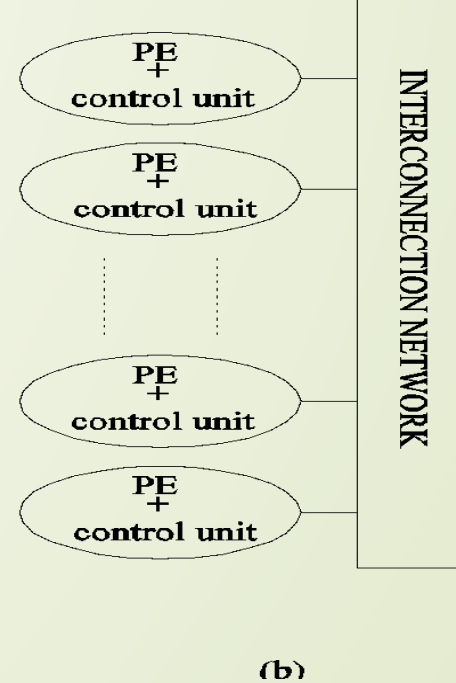


Flynn's Taxonomy

PE: Processing Element



(a)



(b)

A typical SIMD architecture (a) and a typical MIMD architecture (b).
CS3006 - Spring 2022

SIMD-MIMD Comparison

- ? SIMD computers require less hardware than MIMD computers (single control unit).
- ? However, since SIMD processors are specially designed, they tend to be expensive and have long design cycles.
- ? Not all applications are naturally suited to SIMD processors.
- ? In contrast, platforms supporting the SPMD (Same Program Multiple Data) paradigm can be built from inexpensive off-the-shelf components with relatively little effort in a short amount of time.
 - ? The Term SPMD is close variant of MIMD



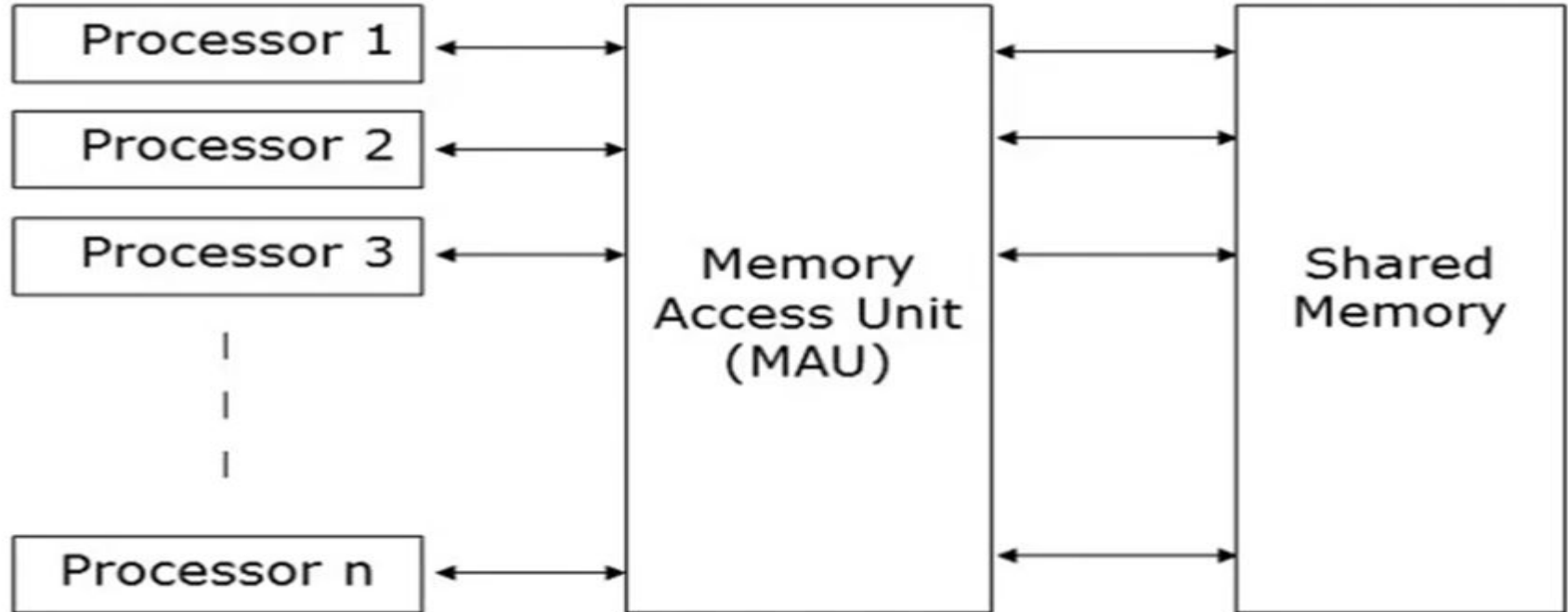
Physical Organization of Parallel Platforms

Architecture of an Ideal Parallel Computer

Parallel Random Access Machine (PRAM)

- ? An extension to ideal sequential model: random access machine (RAM)
- ? PRAMs consist of p processors
- ? A global memory
 - ? Unbounded size
 - ? Uniformly accessible to all processors with same address space
- ? Processors share a common clock but may execute different instructions in each cycle.
- ? Based on simultaneous memory access mechanisms, PRAM can further be classified.

Graphical representation of PRAM:



Architecture of an Ideal Parallel Computer

Parallel Random Access Machine (PRAM)

? PRAMs can be divided into four subclasses.

1. **Exclusive-read, exclusive-write (EREW) PRAM**

? No two processors can perform read/write operations concurrently

? Weakest PRAM model, provides minimum memory access concurrency

2. **Concurrent-read, exclusive-write (CREW) PRAM**

? All processors can read concurrently but can't write at same time

? Multiple write accesses to a memory location are serialized

3. **Exclusive-read, concurrent-write (ERCW) PRAM**

CS3006 - Spring 2022
No two processors can perform read operations concurrently, but can write

4. **Concurrent-read, concurrent-write (CRCW) PRAM**

? Most powerful PRAM model

Architecture of an Ideal Parallel Computer

Parallel Random Access Machine (PRAM)

- ? Concurrent reads do not create any semantic inconsistencies
- ? But, What about concurrent write?
- ? Need of an arbitration(mediation) mechanism to resolve concurrent write access

Architecture of an Ideal Parallel Computer

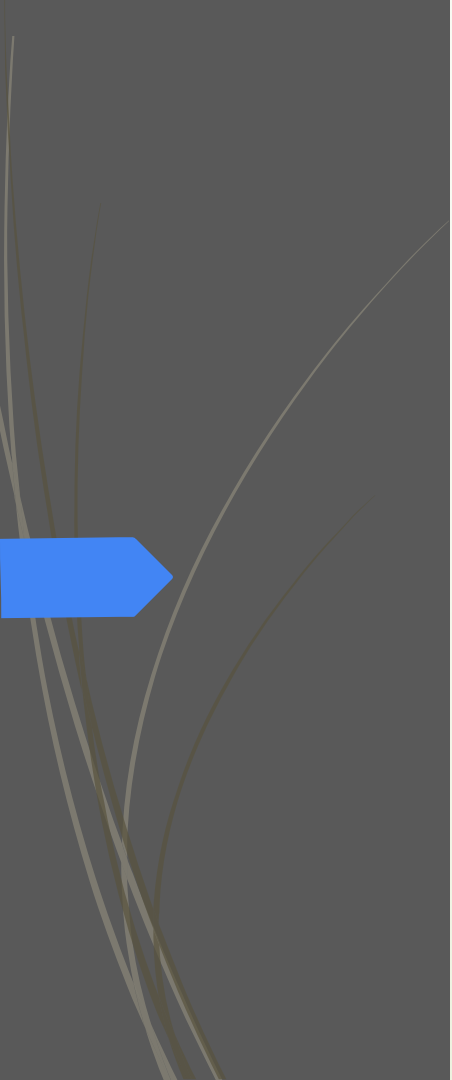
Parallel Random Access Machine (PRAM)

- ? Mostly used arbitration protocols: -
 - ? **Common:** write only if all values that the processors are attempting to write are identical
 - ? **Arbitrary:** write the data from a randomly selected processor and ignore the rest.
 - ? **Priority:** follow a predetermined priority order. Processor with highest priority succeeds and the rest fail.
 - ? **Sum:** Write the sum of the data items in all the write requests. The sum-based write conflict resolution model can be extended for any of the associative operators, that is defined for data being written .

Architecture of an Ideal Parallel Computer

Physical Complexity of an Ideal Parallel Computer

- ? Processors and memories are connected via switches.
- ? Since these switches must operate in $O(1)$ time at the level of words, for a system of p processors and m words, the switch complexity is $O(mp)$.
- ? Clearly, for meaningful values of p and m , a true PRAM is not realizable.



Communication Costs in Parallel Machines

Communication Costs in Parallel Machines

- ? Along with **idling** (doing nothing) and **contention** (conflict e.g., resource allocation), **communication** is a major overhead in parallel programs.
- ? The communication cost is usually dependent on a number of features including the following:
 - ? Programming model for communication
 - ? Network topology
 - ? Data handling and routing
 - ? Associated network protocols
- ? Usually, distributed systems suffer from major communication overheads.

Message Passing Costs in Parallel Computers

- ? The total time to transfer a message over a network comprises of the following:
 - ? **Startup time (t_s):** Time spent at sending and receiving nodes (preparing the message[adding headers, trailers, and parity information] , executing the routing algorithm, establishing interface between node and router, etc.).
 - ? **Per-hop time (t_h):** This time is a function of number of hops (steps) and includes factors such as switch latencies, network delays, etc.
 - ? Also known as **node latency**.
 - ? **Per-word transfer time (t_w):** This time includes all overheads that are determined by the length of the message. This includes bandwidth of links, and buffering overheads, etc.

Message Passing Costs in Parallel Computers

Store-and-Forward Routing

- ? A message traversing multiple hops is completely received at an intermediate hop before being forwarded to the next hop.
- ? The total communication cost for a message of size m words to traverse l communication links is

- ? In most platforms, t_h is small and the above expression can be approximated by

$$t_{comm} = t_s + (mt_w + t_h)l.$$

CS3006 - Spring 2022

$$t_{comm} = t_s + mlt_w.$$

Message Passing Costs in Parallel Computers

Packet Routing

- ? Store-and-forward makes poor use of communication resources.
- ? Packet routing breaks messages into packets and pipelines them through the network.
- ? Since packets may take different paths, each packet must carry routing information, error checking, sequencing, and other related header information.

GS3006 - Spring 2022

- ? The total communication time for packet routing is approximated by: $t_{comm} = t_s + t_h l + t_w m.$
- ? Here factor t_w also accounts for overheads in packet headers.

Message Passing Costs in Parallel Computers

Cut-Through Routing

- ? Takes the concept of packet routing to an extreme by further dividing messages into basic units called **flits** or flow control digits.
- ? Since flits are typically small, the header information must be minimized.
- ? This is done by forcing all flits to take the same path, in sequence.
- ? A tracer message first programs all intermediate routers. All flits then take the same route.
- GS3006 - Spring 2022
 - ? Error checks are performed on the entire message, as opposed to flits.
 - ? No sequence numbers are needed.

Message Passing Costs in Parallel Computers

Cut-Through Routing

? The total communication time for cut-through routing is approximated by:

$$t_{comm} = t_s + t_h l + t_w m.$$

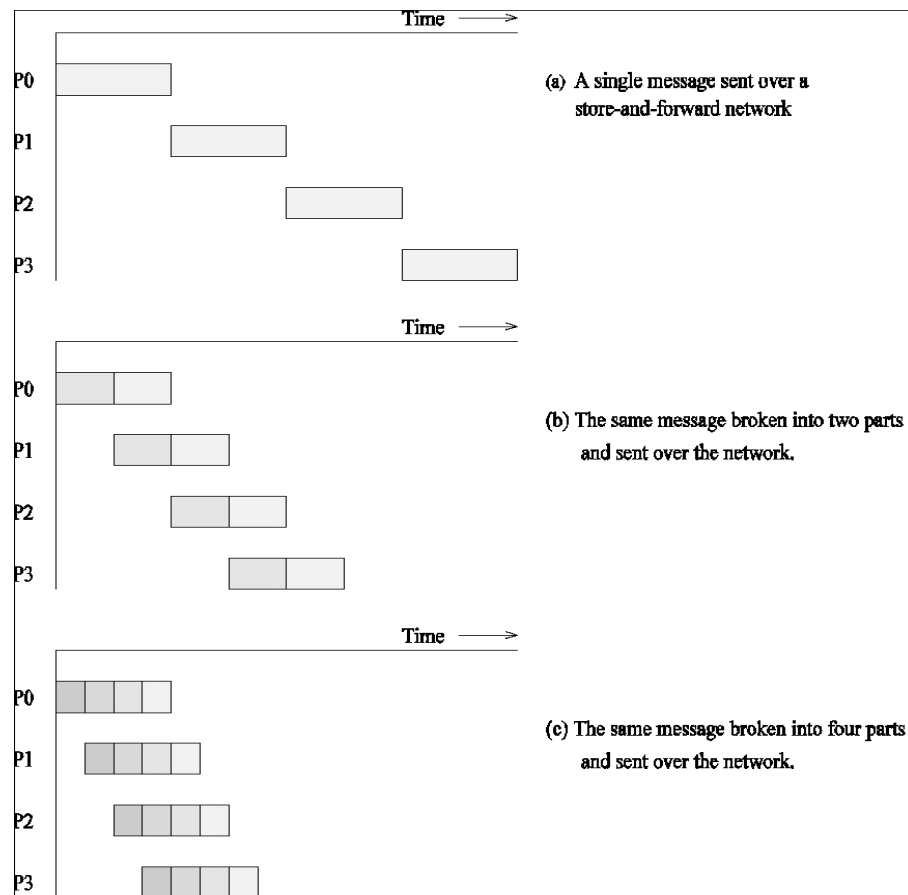
? This is identical to packet routing, however, t_w is typically much smaller.

Message Passing Costs in Parallel Computers

(a) through a store-and-forward communication network;

b) and (c) extending the concept to cut-through routing.

CS3006 - Spring 2022



Message Passing Costs in Parallel Computers

Simplified Cost Model for Communicating Messages

? The cost of communicating a message between two nodes / hops away using cut-through routing is given by



? In this expression, t_h is typically smaller than t_s and t_w . For this reason, the second term in the RHS does not show, particularly, when m is large.

CS3006 - Spring 2022

? For these reasons, we can approximate the cost of message transfer by

$$t_{comm} = t_s + t_w m.$$

Message Passing Costs in Parallel Computers

Simplified Cost Model for Communicating Messages

- ? It is important to note that the original expression for communication time is valid for only **uncongested networks**.
- ? Different communication patterns congest different networks to varying extents.
- ? It is important to **understand and account for** this in the communication time accordingly.

Questions



References

1. Flynn, M., “Some Computer Organizations and Their Effectiveness,” IEEE Transactions on Computers, Vol. C-21, No. 9, September 1972.
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).



Parallel and Distributed Computing

CS3006

Lecture 4

Network Topologies

28th February 2022

Dr. Rana Asif Rehman



Agenda

- ? **A Quick Review**
- ? **Static Interconnection vs Dynamic interconnections**
- ? **Some Basic Interconnections**
- ? **Evaluating Static Interconnections**

CS3006 - Spring 2022

Quick Review to the Previous Lecture



? **Flynn's Taxonomy**

- ? SISD
- ? MISD
- ? SIMD
- ? MIMD

? **PRAM Model**

- ? Types
- ? Arbitration protocols

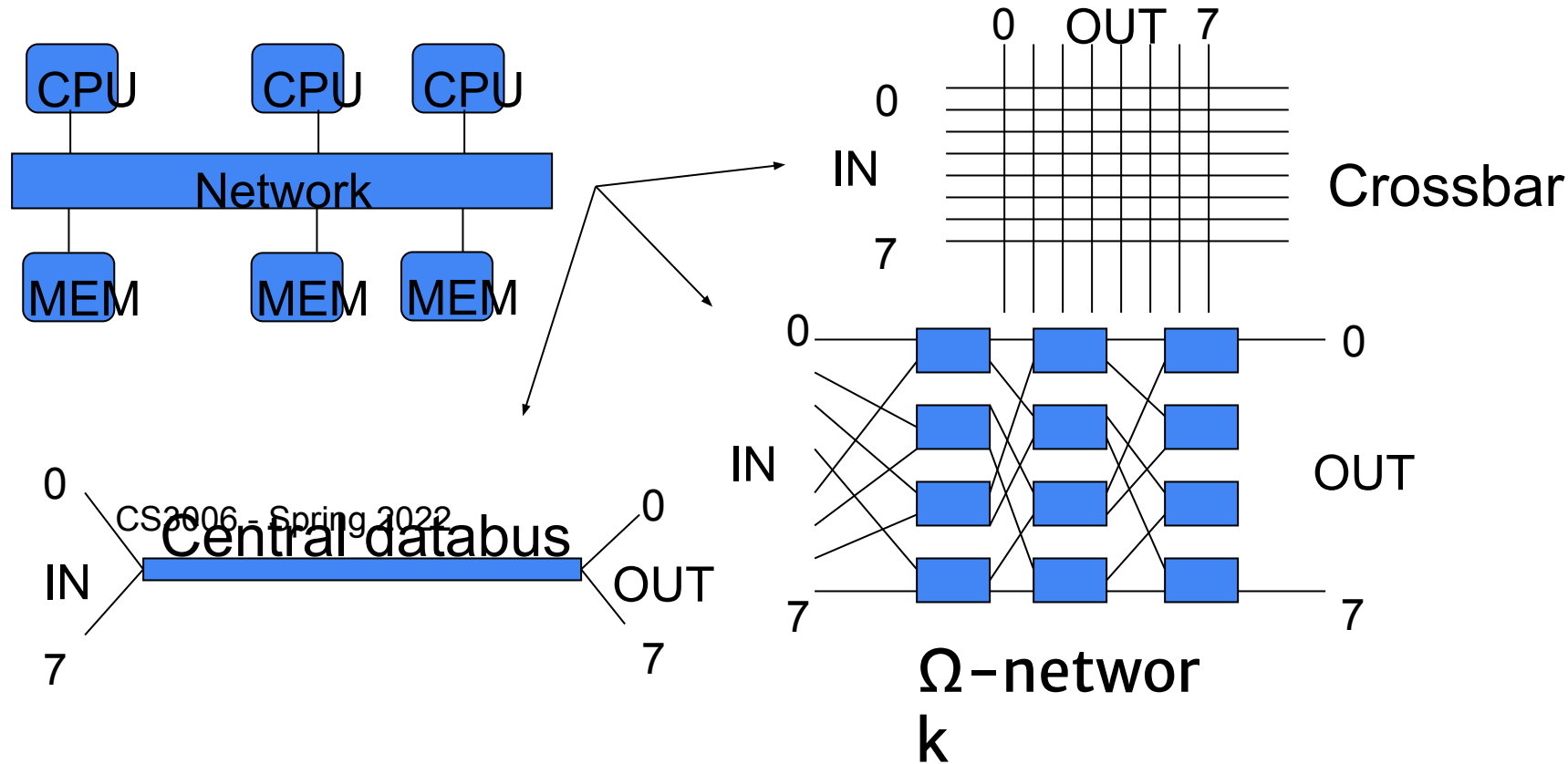
? **Routing techniques and Costs**

CS3006 - Spring 2022

Interconnection Networks

- ? Main problem is how to do interconnections of the CPUs to each other and to the memory
- ? There are three main network topologies available:
 - ? **Crossbar** (n^2 connections – data path without sharing)
 - ? **CS3006 - Spring 2022** **Multi-stages network** ($n \log_2 n$ connections - $\log_2 n$ switching stages and shared on a path)
 - ? **Central databus** (1 connections - n shared)

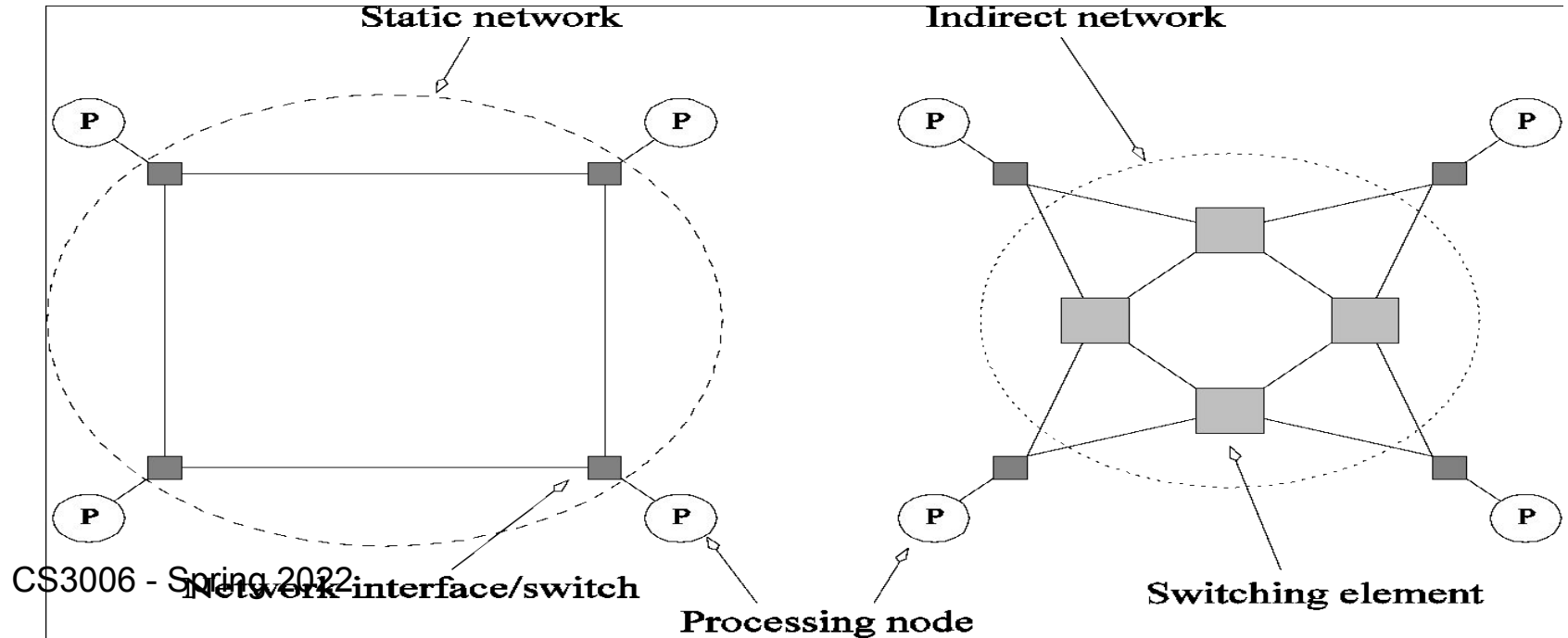
Interconnection Networks



Static vs Dynamic Interconnections

- ? Interconnection networks carry data between processors and to memory.
- ? Interconnects are made of processing elements, switches and links (wires, fiber).
- ? Interconnects are classified as static or dynamic.
- ? **Static** networks consist of point-to-point communication links among processing nodes and are also referred to as *direct* networks.
- ? **Dynamic** networks are built using switches and communication links. Dynamic networks are also referred to as *indirect* networks.

Static vs Dynamic Interconnections

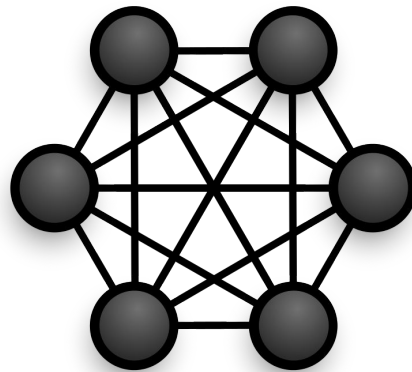


Classification of interconnection networks: (a) a static network; and (b) a dynamic network.

Network Topologies:

Linear Arrays, Meshes, and *k-d* Meshes

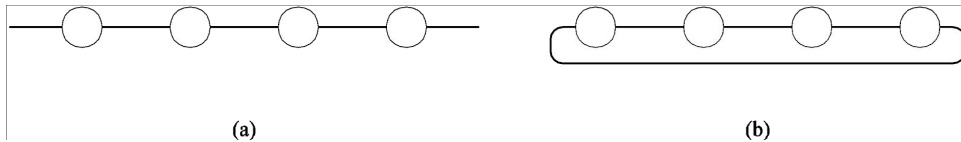
- ? Each processor is connected to every other processor (Complete connected network).
- ? The number of links in the network scales as $O(p^2)$.
- ? While the performance scales very well, the hardware complexity is not realizable for large values of p .
- ? Star connected networks



Network Topologies:

Linear Arrays, Meshes, and *k-d* Meshes

- ? In a linear array, each node has two neighbors, one to its left and one to its right.
- ? If the nodes at either end are connected, we refer to it as a 1-D torus or a ring.



CS3006 - Spring 2022

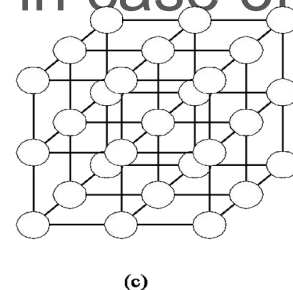
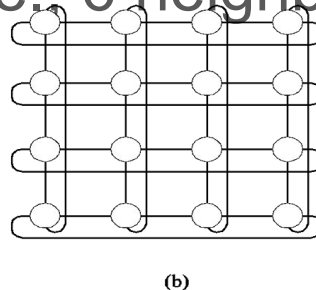
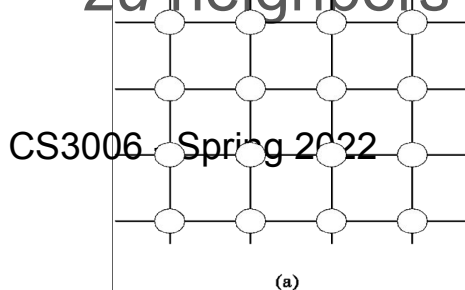
Linear arrays: (a) with no wraparound links; (b) with wraparound link.

Network Topologies:

Linear Arrays, Meshes, and k - d Meshes

Mesh

- ? A generalization has nodes with 4 neighbors, to the north, south, east, and west.
- ? A further generalization to d dimensions has nodes with $2d$ neighbors (i.e., 6 neighbors in case of 3d cube).



Two and three dimensional meshes: (a) 2-D mesh with no wraparound; (b) 2-D mesh with wraparound link (2-D torus); and (c) a 3-D mesh with no wraparound

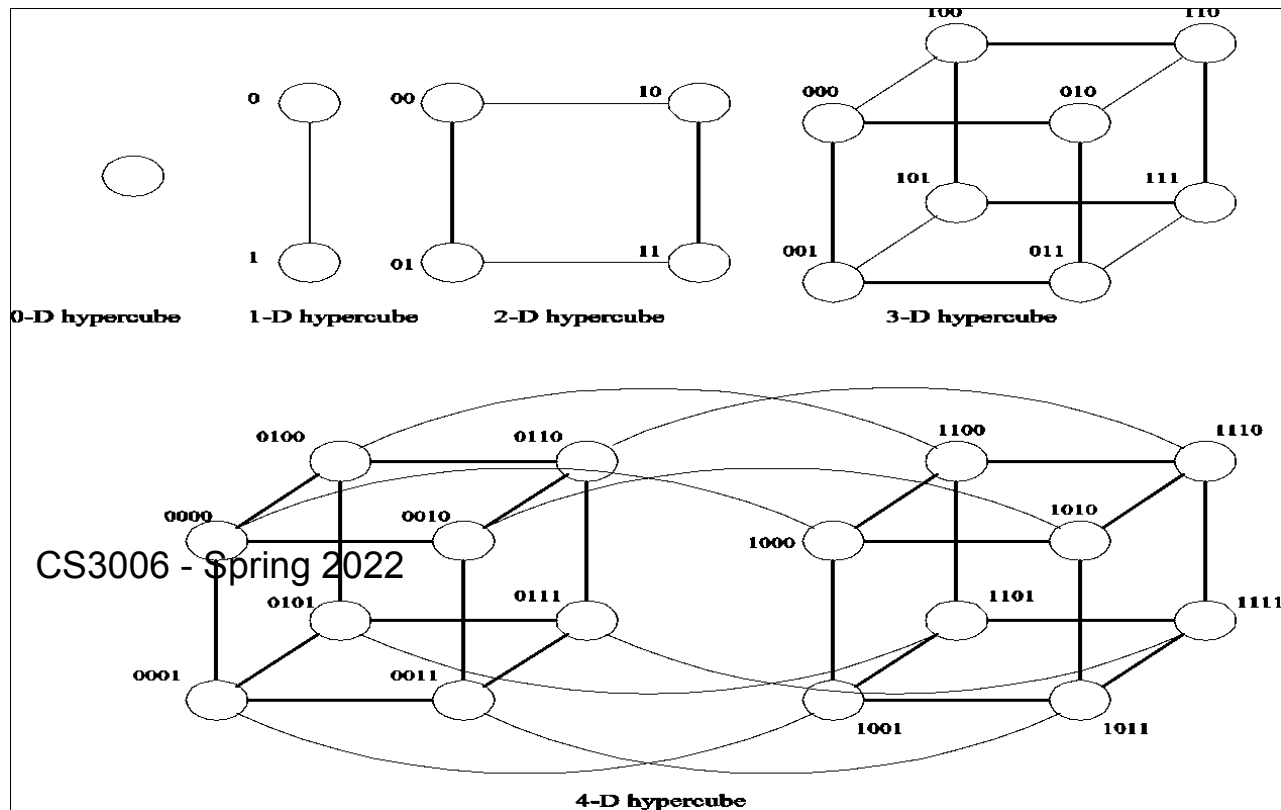
Network Topologies:

Linear Arrays, Meshes, and k - d Meshes

Hypercube

- ? The hypercube has two nodes along each dimension except $0d$ hypercube.
- ? $d = \log p$ (*dimensions = log(nodes)*)
- ? The distance between any two nodes is at most $\log p$.
- ? Each node has $\log p$ neighbors.
- ? The distance between two nodes is given by the number of bit positions at which the two nodes differ.
- ? Rule of thumb is: “ d -dimensional hypercube can be constructed by connecting corresponding nodes of two $(d-1)$ -dimensional hypercubes”

Network Topologies: Linear Arrays, Meshes, and k - d Meshes





Network Topologies:

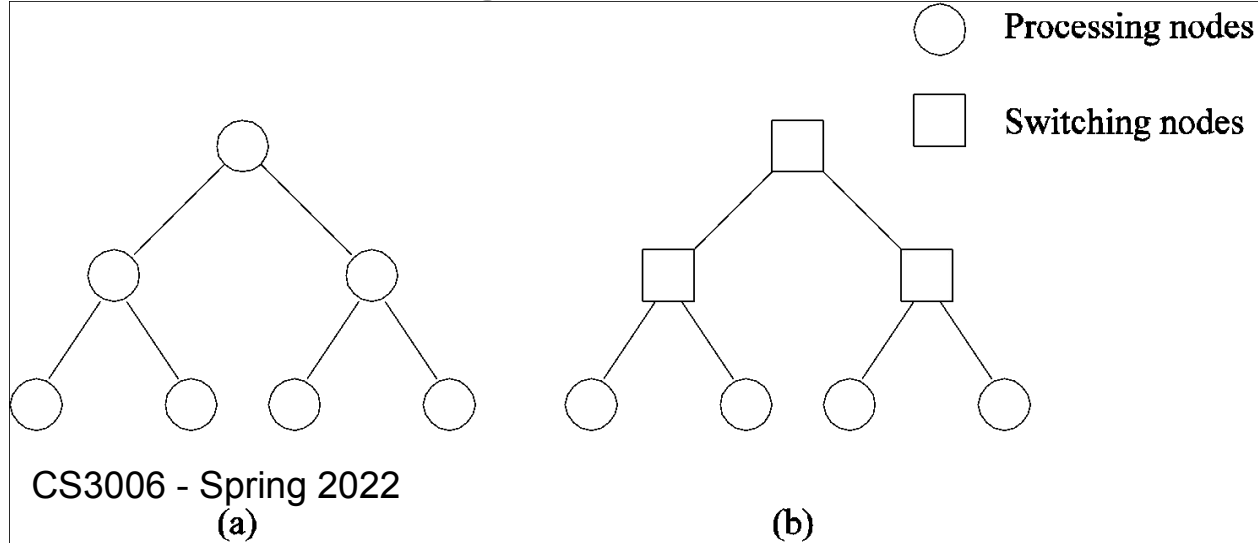
Tree based Networks

- ? A tree network is one in which there is one path between any pair of nodes
- ? Linear arrays and star-connected networks are special cases of tree-based networks
- ? In static tree network, each node represent a processing element
- ? In dynamic tree network, leaf nodes represent processing element while internal nodes are switching elements.
- ? The source node sends the message up the tree until it reaches the node at the root of the smallest subtree containing both the source and destination nodes.

Network Topologies:

Tree based Networks

Complete Binary Tree



Complete binary tree networks: (a) a static tree network; and (b) a dynamic tree network.



Network Topologies:

Tree based Networks

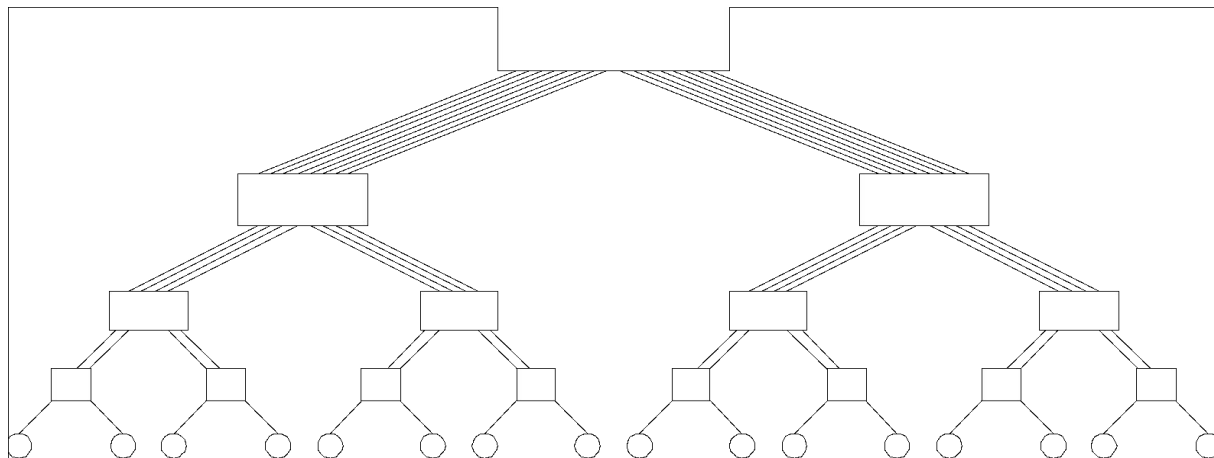
Properties of Complete Binary Tree Network

- ? The distance between any two nodes is no more than $2\log p$.
 - ? Links higher up the tree potentially carry more traffic than those at the lower levels.
 - ? For this reason, a variant called a fat-tree, fattens the links as we go up the tree.
- CS3006 - Spring 2022
- ? Trees can be laid out in 2D with no wire crossings. This is an attractive property of trees.

Network Topologies:

Tree based Networks

Properties of Complete Binary Tree Network



CS3006 - Spring 2022

A fat tree network of 16 processing nodes.

Evaluating Static Interconnections

The parameters to evaluate a static interconnection:-

- ? **Cost:** Usually depends on number of links for communication. E.g., cost for linear array is $p-1$.
 - ? *Lower values are favorable*
- ? **Diameter:** The shortest distance between the farthest two nodes in the network. The diameter of a linear array is $p - 1$.
 - ? *Lower values are favorable*
- ? **Bisection Width:** The minimum number of wires you must cut to divide the network into two equal parts. The bisection width of a linear array is 1.
 - ? What it tells about performance of a topology?

Evaluating Static Interconnections

The parameters to evaluate a static interconnection:-

? Arc-connectivity: *The minimum number of arcs or links that must be removed from the network, to break the network into two disconnected networks*

? Higher value are desirable

? It is minimum number of the links that must be cut to separate the single node from the network

? Higher values means, that incase of link failure there are multiple other routes to the node.

? Arc-connectivity of linear array is 1 and 2 for ring.

Evaluating Static Interconnections

Network	Diameter	Bisection Width	Arc Connectivity	Cost (No. of links)
Completely-connected	1	$p^2/4$	$p - 1$	$p(p - 1)/2$
Star	2	1	1	$p - 1$
Complete binary tree	$2 \log((p + 1)/2)$	1	1	$p - 1$
Linear array	$p - 1$	1	1	$p - 1$
2-D mesh, no wraparound	$2(\sqrt{p} - 1)$	\sqrt{p}	2	$2(p - \sqrt{p})$
2-D wraparound mesh	$2\lfloor \sqrt{p}/2 \rfloor$	$2\sqrt{p}$	4	$2p$
Hypercube	$\log p$	$p/2$	$\log p$	$(p \log p)/2$

Questions



References

1. Flynn, M., “Some Computer Organizations and Their Effectiveness,” IEEE Transactions on Computers, Vol. C-21, No. 9, September 1972.
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).

Cache Coherence and snooping

- ? In a snooping system, all caches on the bus monitor (or snoop) the bus to determine if they have a copy of the block of data that is requested on the bus.
- ? Every cache has a copy of the sharing status of every block of physical memory it has.

Snooping Protocol Types

- ? Write-invalidate (mostly used)
 - ? The processor that is writing data causes copies in the caches of all other processors in the system to be rendered **invalid** before it changes its local copy.
- ? Write-update
 - ? The processor that is writing the data broadcasts the new data over the bus
 - ? All caches that contain copies of the data are then updated



Parallel and Distributed Computing

CS3006

Lecture 5

Parallel Algorithm Design Life Cycle

7th March 2022

Dr. Rana Asif Rehman

Agenda

- ? A Quick Review
- ? Parallel Algorithm Design Life Cycle
- ? Tasks, Decomposition, and Task-dependency graphs
- ? Granularity
 - ? Fine-grained
 - ? Coarse-grained
- ? Concurrency
 - ? Max-degree of concurrency
 - ? Critical path length
 - ? Average-degree of concurrency
- ? Task-interaction Diagrams
 - ? Processes and mapping

CS3006 - Spring 2022

Quick Review to the Previous Lecture

? Static vs Dynamic Interconnections

? Network Topologies

? Linear array

? Star

? Mesh

? Tree

? Fully-connected

? Hypercube

? Evaluating Static interconnections

? Cost

? Diameter

? Bisection-width

? Arc connectivity

CS3006 - Spring 2022



Principles of Parallel Algorithm Design

CS3006 - Spring 2022

Principles of Parallel Algorithm Design

Steps in Parallel Algorithm Design

1. **Identification:** Identifying portions of the work that can be performed concurrently.
 - ? Work-units are also known as tasks
 - ? E.g., Initializing two mega-arrays are two tasks and can be performed in parallel
2. **Mapping:** The process of mapping concurrent pieces of the work or tasks onto multiple processes running in parallel.
 - ? Multiple processes can be physically mapped on a single processor.

Principles of Parallel Algorithm Design

Steps in Parallel Algorithm Design

3. **Data Partitioning:** Distributing the input, output, and intermediate data associated with the program.
 - ? One way is to copy whole data at each processing node
 - ? Memory challenges for huge-size problems
 - ? Other way is to give fragments of data to each processing node
 - ? Communication overheads
4. **Defining Access Protocol:** Managing accesses to data shared by multiple processors (i.e., managing communication).
5. **Synchronizing** the processors at various stages of the parallel program execution.

Principles of Parallel Algorithm Design

? Decomposition:

- ? The process of dividing a computation into smaller parts, some or all of which may potentially be executed in parallel.

? *Tasks*

- ? **Programmer-defined units of computation** into which the main computation is subdivided by means of decomposition
- ? Tasks can be of **arbitrary size**, but once defined, they are regarded as **indivisible units of computation**.

CS3006 - Spring 2022

- ? The tasks into which a problem is decomposed may not all be of the same size

- ? Simultaneous execution of multiple tasks is the key to reducing the time required to solve the entire problem.

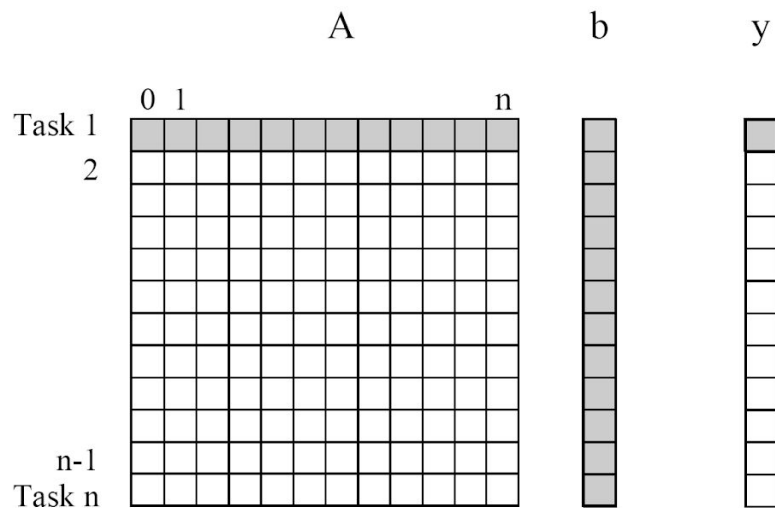


Figure 3.1 Decomposition of dense matrix-vector multiplication into n tasks, where n is the number of rows in the matrix. The portions of the matrix and the input and output vectors accessed by Task 1 are highlighted.

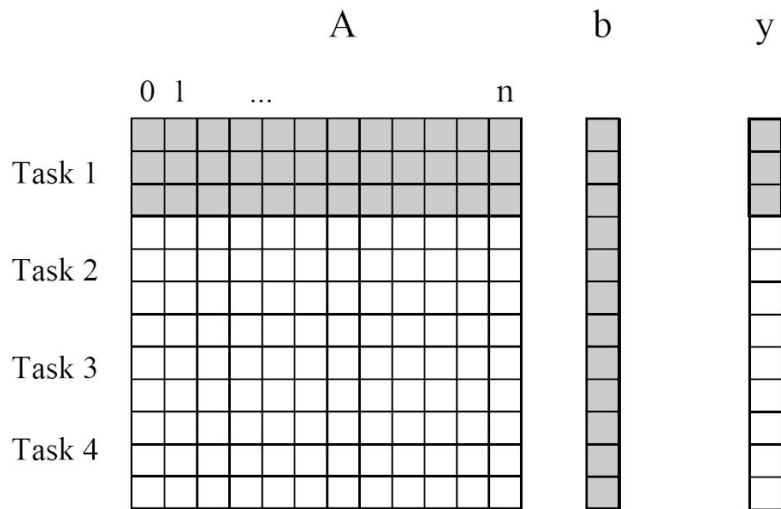


Figure 3.4 Decomposition of dense matrix-vector multiplication into four tasks. The portions of the matrix and the input and output vectors accessed by Task 1 are highlighted.

Principles of Parallel Algorithm Design

Task-Dependency Graph

- ? The tasks in the previous examples are independent and can be performed in any sequence.
- ? In most of the problems, there exist some sort of dependencies between the tasks.
- ? An abstraction used to express such **dependencies** among tasks and their **relative order of execution** is known as a **task-dependency graph**
- ? It is a **directed acyclic graph** in which node are tasks and the directed edges indicate the dependencies between them
- ? The task corresponding to a node can be executed when all tasks connected to this node by incoming edges have completed.

ID#	Model	Year	Color	Dealer	Price
4523	Civic	2002	Blue	MN	\$18,000
3476	Corolla	1999	White	IL	\$15,000
7623	Camry	2001	Green	NY	\$21,000
9834	Prius	2001	Green	CA	\$18,000
6734	Civic	2001	White	OR	\$17,000
5342	Altima	2001	Green	FL	\$19,000
3845	Maxima	2001	Blue	NY	\$22,000
8354	Accord	2000	Green	VT	\$18,000
4395	Civic	2001	Red	CA	\$17,000
7352	Civic	2002	Red	WA	\$18,000

Table 3.1 A database storing information about used vehicles.

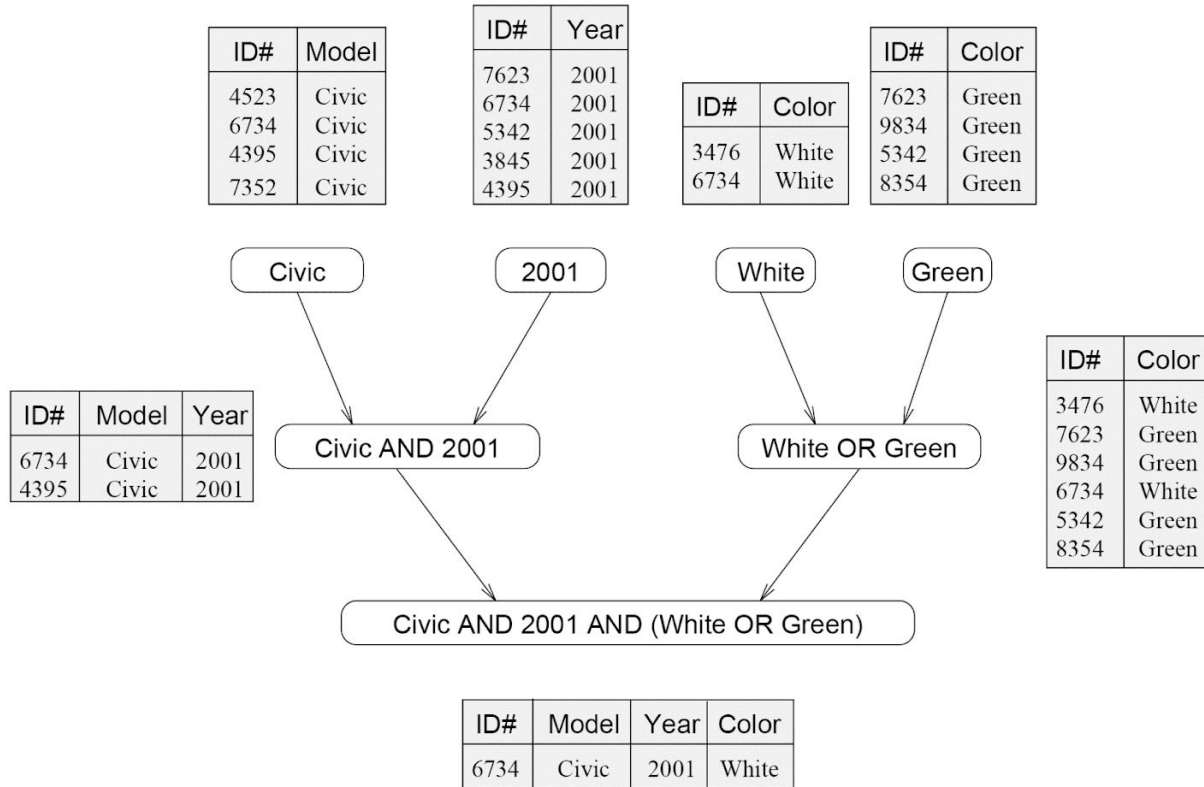


Figure 3.2 The different tables and their dependencies in a query processing operation.

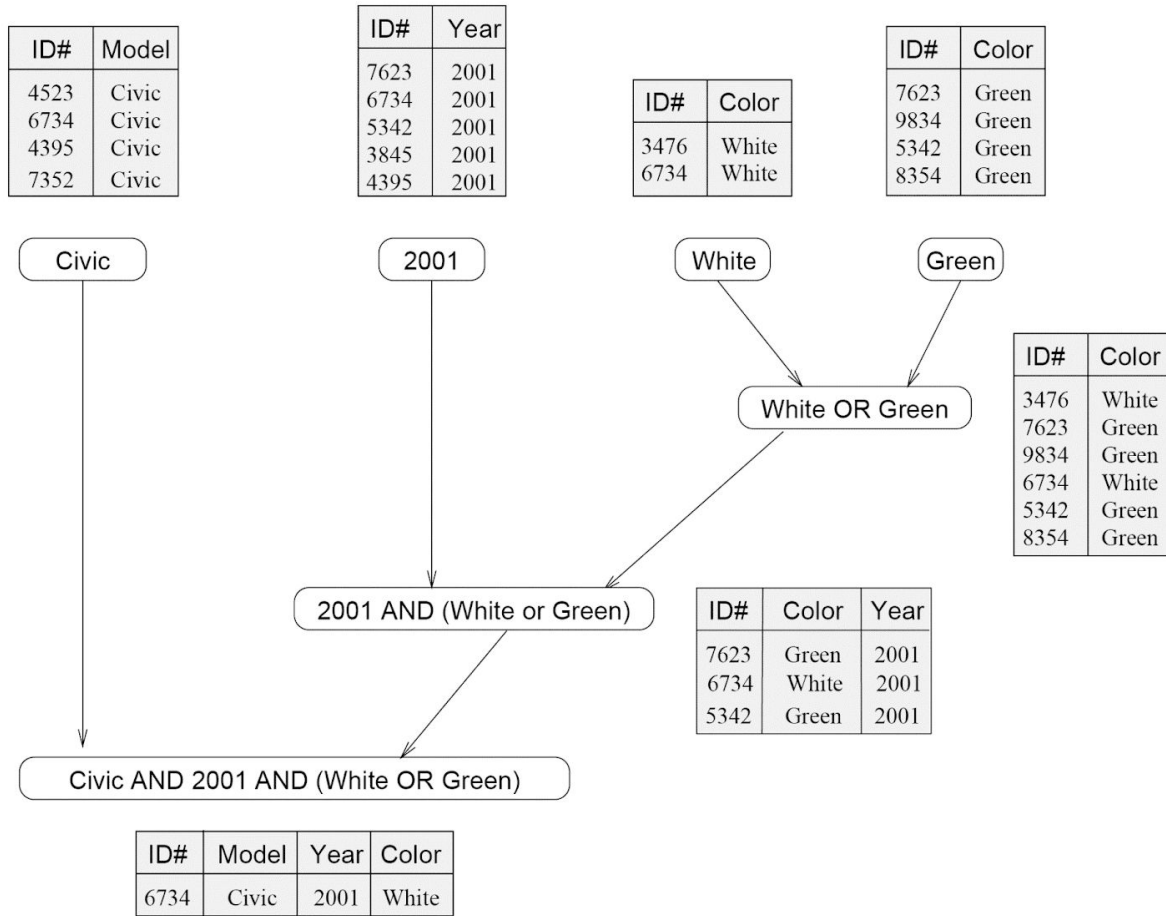


Figure 3.3 An alternate data-dependency graph for the query processing operation.

Principles of Parallel Algorithm Design

Granularity

- ? The **number and sizes** of tasks into which a problem is decomposed determines the ***granularity*** of the decomposition
 - ? A decomposition into a large number of small tasks is called ***fine-grained***
 - ? A decomposition into a small number of large tasks is called ***coarse-grained***
- ? For matrix-vector multiplication Figure 3.1 would usually be considered fine-grained
- ? Figure 3.4 shows a coarse-grained decomposition as each task computes $n/4$ of the entries of the output vector of length n

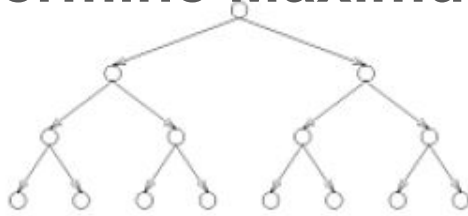
principles of Parallel Algorithm Design

Maximum Degree of Concurrency

- ? The maximum number of tasks that can be executed simultaneously in a parallel program at any given time is known as its *maximum degree of concurrency*
- ? Usually, it is always less than total number of tasks due to dependencies.
- ? E.g., max-degree of concurrency in the task-graphs of Figures 3.2 and 3.3 is 4.
- ? CS3006 - Spring 2022 **Rule of thumb:** For task-dependency graphs that are trees, the maximum degree of concurrency **is always equal to the number of leaves in the tree**

principles of Parallel Algorithm Design

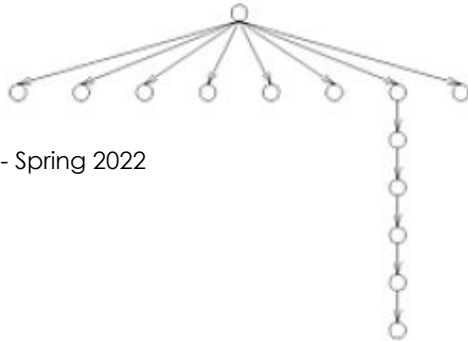
Determine Maximum Degree of Concurrency?



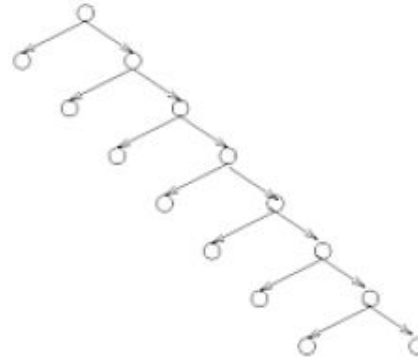
(a)



(b)



(c)



(d)

principles of Parallel Algorithm Design

Average Degree of Concurrency

- ? A relatively better measure for the performance of a parallel program
- ? The average number of tasks that can run concurrently over the entire duration of execution of the program
- ? The ratio of the ***total amount of work*** to the ***critical-path length***
 - ? So, what is the critical path in the graph?

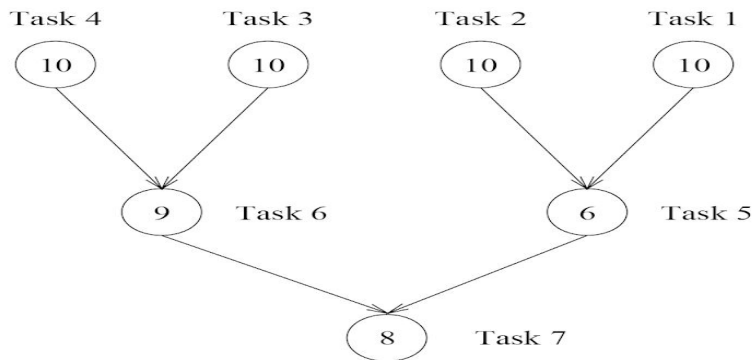
principles of Parallel Algorithm Design

Average Degree of Concurrency

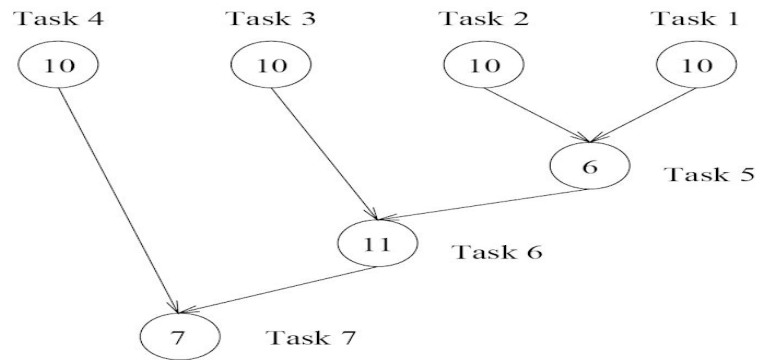
- ? **Critical Path:** The longest directed path between any pair of start and finish nodes is known as the critical path.
- ? **Critical Path Length:** The sum of the weights of nodes along this path
 - ? the weight of a node is the size or the amount of work associated with the corresponding task.
- ? A shorter critical path favors a higher average-degree of concurrency.
- ? Both, maximum and average degree of concurrency increases as tasks become smaller(finer)

principles of Parallel Algorithm Design

Average Degree of Concurrency



(a)



(b)

CS3006 Spring 2022

Figure 3.5 Abstractions of the task graphs of Figures 3.2 and 3.3, respectively.

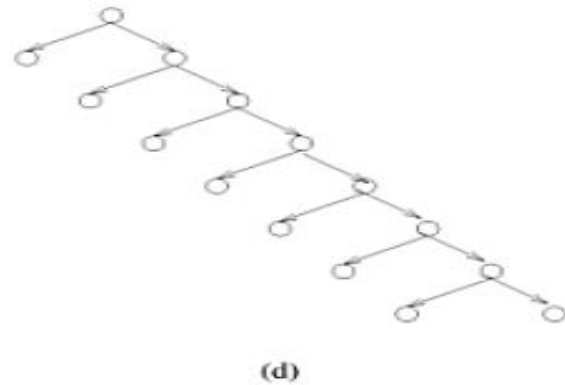
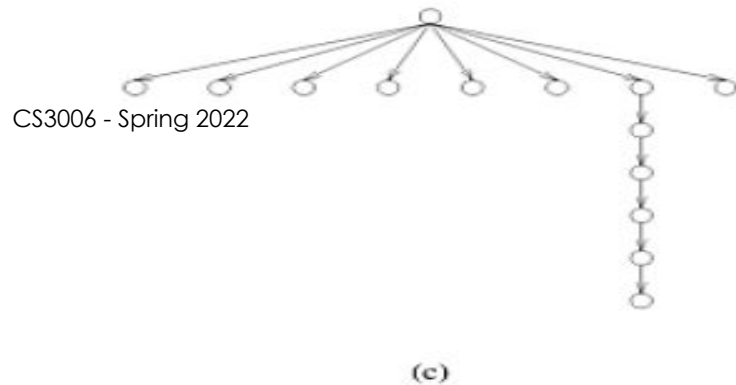
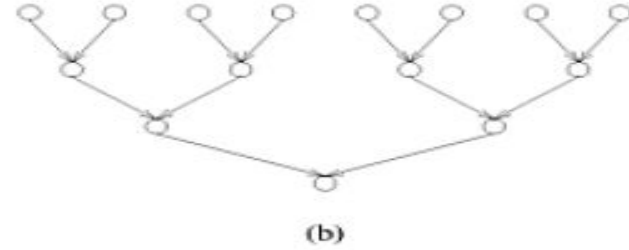
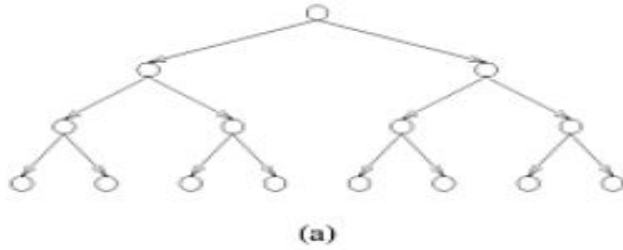
Critical path lengths: 27 and 34

Total amount of work: 63 and 64

Average degree of concurrency: 2.33 and 1.88

principles of Parallel Algorithm Design

Determine critical path length and average-concurrency?



principles of Parallel Algorithm Design

Task Interact Graph

- ? Depicts pattern of interaction between the tasks
- ? Dependency graphs only show that how output of first task becomes input to the next level task.
- ? But how the tasks interact with each other to access distributed data is only depicted by task interaction graphs
- ? The nodes in a task-interaction graph represent tasks
- ? The edges connect tasks that interact with each other

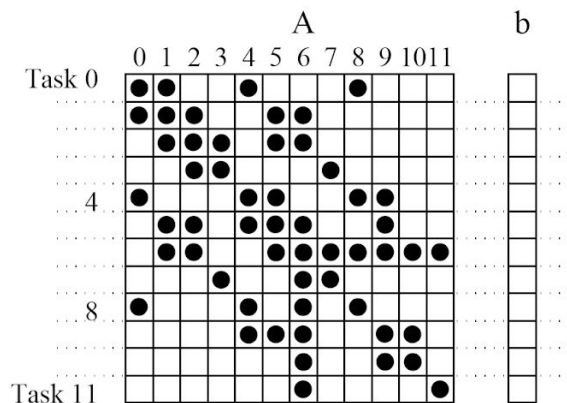
principles of Parallel Algorithm Design

Task Interact Graph

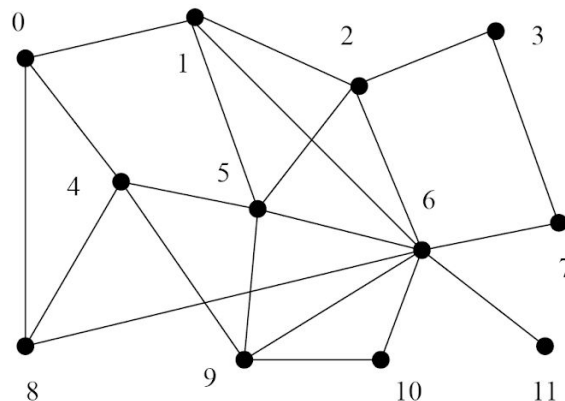
- ? The edges in a task interaction graph are usually **undirected**
 - ? but directed edges can be used to indicate the direction of flow of data, if it is unidirectional.
- ? The edge-set of a task-interaction graph is usually a **superset** of the edge-set of the task-dependency graph
- ? In database query processing example, the task-interaction graph is the **same** as the task-dependency graph.

principles of Parallel Algorithm Design

Task Interact Graph (Sparse-matrix multiplication)



CS3006 - Spring 2022
(a)



(b)

Figure 3.6 A decomposition for sparse matrix-vector multiplication and the corresponding task-interaction graph. In the decomposition Task i computes $\sum_{0 \leq j \leq 11, A[i,j] \neq 0} A[i,j] \cdot b[j]$.

principles of Parallel Algorithm Design

Processes and Mapping

- ? Logical processing or computing agent that performs tasks is called **process**.
- ? The mechanism by which tasks are assigned to processes for execution is called **mapping**.
- ? Multiple tasks can be mapped on a single process
- ? Independent task should be mapped onto different processes
- ? Map tasks with high mutual-interactions onto a single process

principles of Parallel Algorithm Design

Processes and Mapping

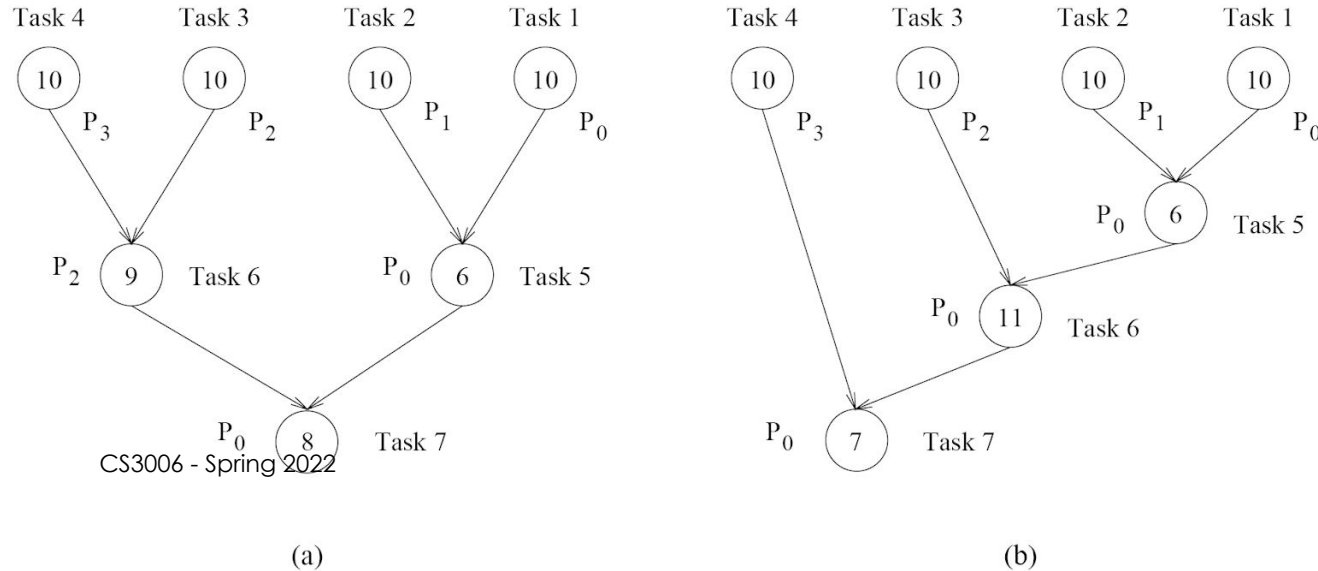


Figure 3.7 Mappings of the task graphs of Figure 3.5 onto four processes.

principles of Parallel Algorithm Design

Processes and Processors

- ? **Processes** are logical computing agents that perform tasks
- ? **Processors** are the hardware units that physically perform computations
- ? Depending on the problem, multiple processes can be mapped on a single processor
- ? But, in most of the cases, there is one-to-one correspondence between processors and processes
- ? So, we assume that there are as many processes as the number of physical CPUs on the parallel computer

Questions



References

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