



Parallel and Distributed Computing

Lecture 2

Amdahl's Law

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

Amdahl's Law

- Amdahl's was formulized 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?
- ➡ What you have to do for further speedup?

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

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= to 99 do

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

Endfor

- Here same operation addition is being performed on first 100 of 'b' and 'c'
- All 100 iterations of the loop could be executed simultaneously.

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

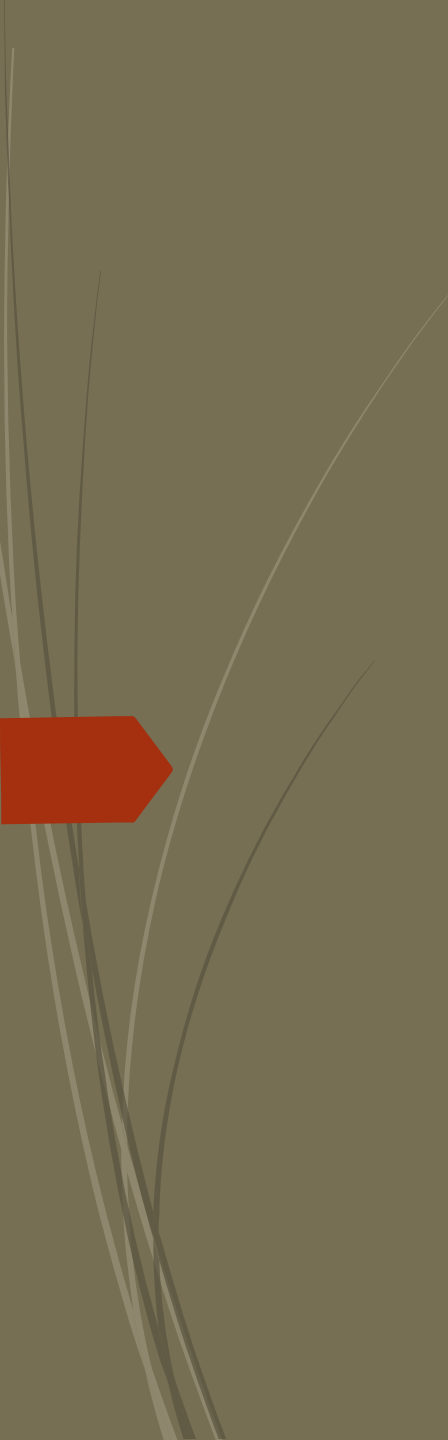
Types of Parallelism

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

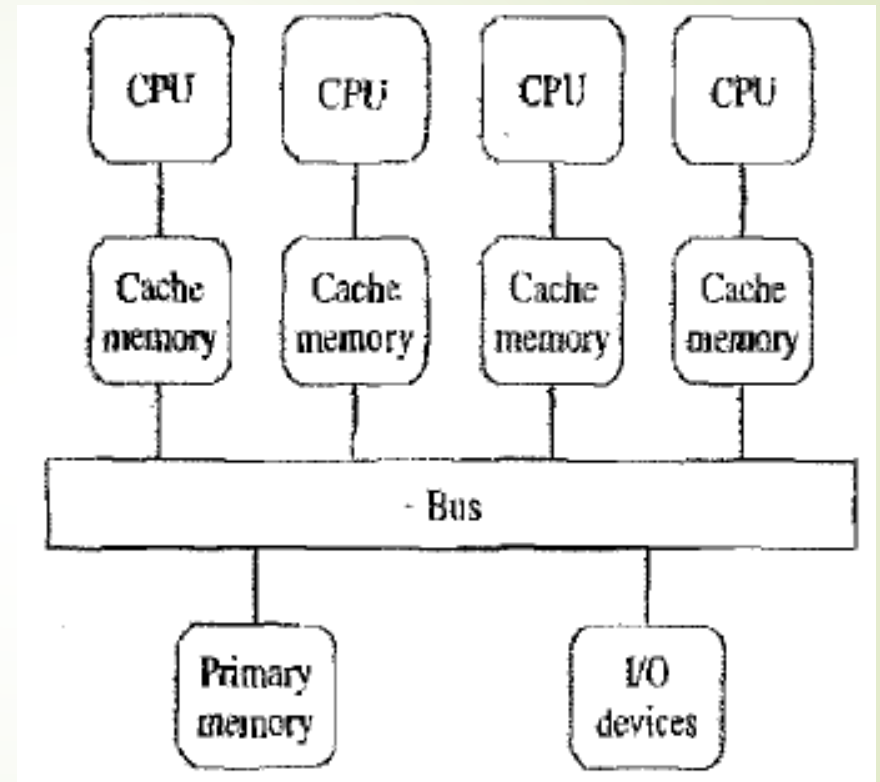
Multi-Processor

- 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

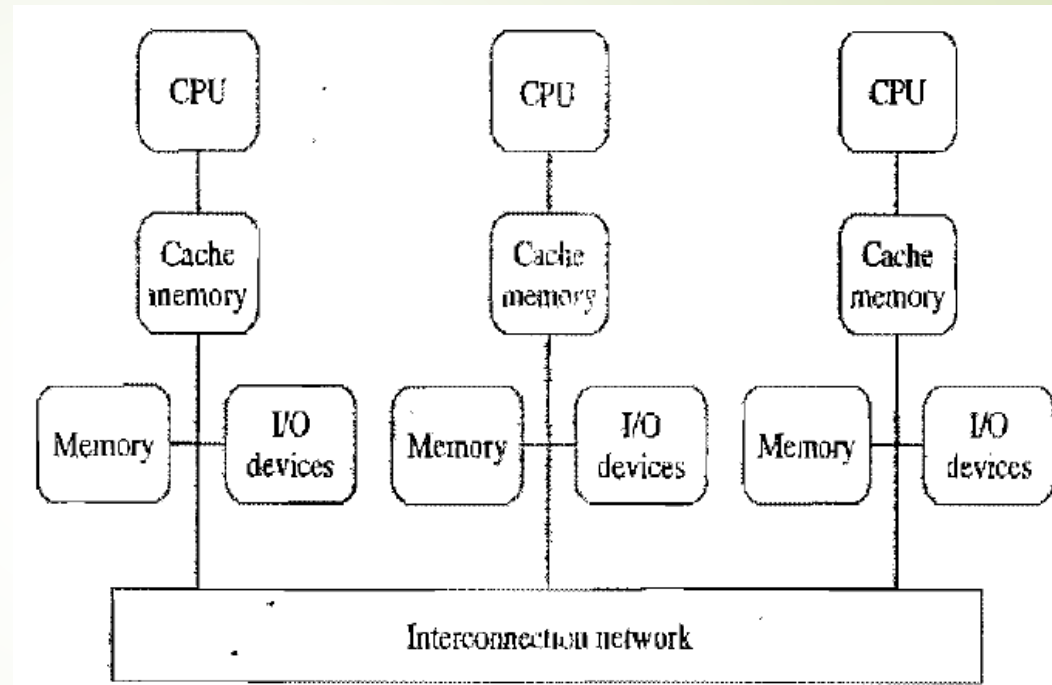
- Additional CPUs are attached to the system bus, and all the processors share the same primary memory
- All the memory is at one place and has the same access time from every processor
- Also known to as **UMA** (Uniform Memory Access) multi-processor or **SMP** (symmetrical Multi-processor)



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



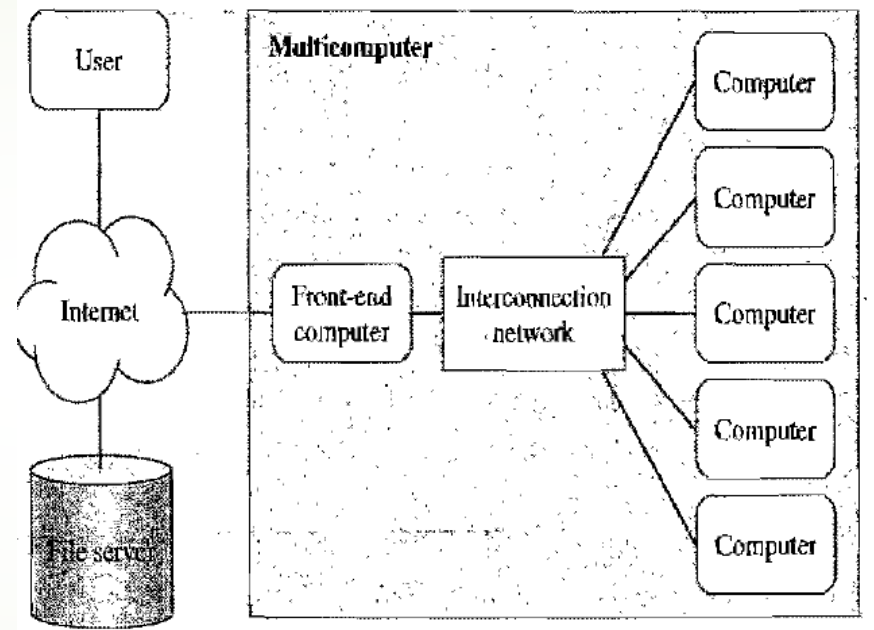
Multi-Computer

- 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

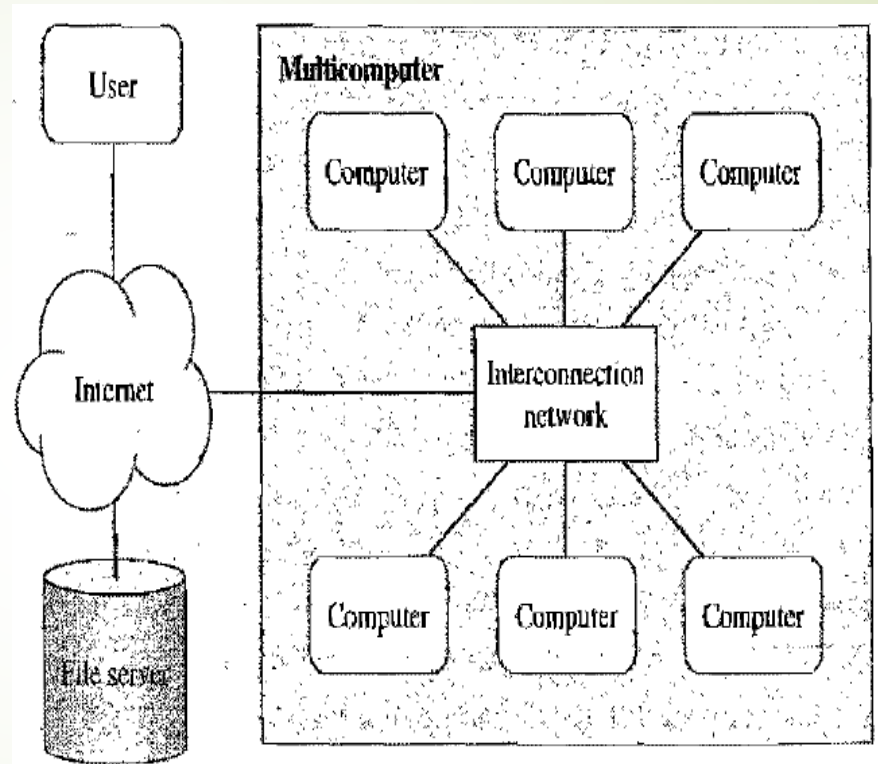
- A front-end computer that interacts with users and I/O devices
- The back-end processors are dedicatedly used for “number crunching”
- Front-end computer executes a full, multiprogrammed OS and provides all functions needed for program development
- The backends are reserved for executing parallel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

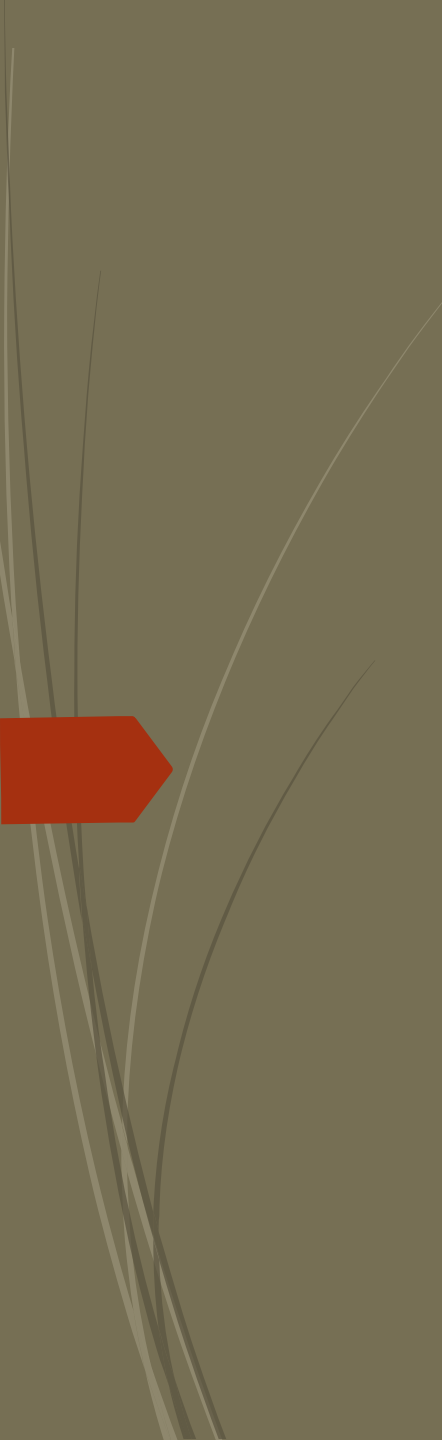


Multi-Computer

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



