# Parallel and Distributed Computing
# CS3006 (BDS-6A)
# Lecture 02

Instructor: Dr. Syed Mohammad Irteza

Assistant Professor, Department of Computer Science, FAST

02 February, 2023

# Administrative Information

- Google Classroom:
- https://classroom.google.com/c/NTg2ODY0MTcyNjA5
  - Class code: 22a4ago

# Computing v. 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)

# Previous Lecture

- Motivating Parallelism
  - Moore’s Law
  - Memory/Disk Speed Argument
  - Data Communication Argument
- Distributed Computing v. Distributed Systems

# Parallel and 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 PDC

***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 PDC

***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 natural hazards like the extreme cyclones (storms).
- Flood prediction

# Practical Applications of PDC

***Commercial Applications***:

- Some of the largest parallel computers power 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 PDC

***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*.

# Moving on………

- How can we quantify the possible gains from parallelization?
  - Amdahl's Law is a good starting point

# 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

$$\mathbf{Speedup(p)} = \frac{\mathbf{1}}{\boldsymbol{F} + \frac{\mathbf{1} - \boldsymbol{F}}{\boldsymbol{p}}}$$

- 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)*** *will* be the parallel time for the same algorithm over *p* processors.

***Then speedup can be calculated using:-***

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

- T(p) can be calculated as:

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

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

# Amdahl's Law

**Derivation**

- Again

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

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

- What if you have an infinite number of processors?
- What do you have to do for further speedup?

# Amdahl's Law

- **Example 1:** Suppose 70% of a sequential algorithm is the parallelizable portion. The remaining part must be calculated sequentially. Calculate maximum theoretical speedup for parallel variant of this algorithm using
  - 4 processors, and
  - Infinite processors.

- F= $0.30$ and 1-F= $0.70$ use Amdahl's law to calculate theoretical speedups.

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

# Amdahl's Law

- **Example 1:** Suppose 70% of a sequential algorithm is the parallelizable portion. The remaining part must be calculated sequentially. Calculate maximum theoretical speedup for parallel variant of this algorithm using
  - 4 processors, and
  - Infinite processors.

- F= $0.30$ and 1-F= $0.70$ use Amdahl's law to calculate theoretical speedups.

Speedup(p=4) = $\frac{1}{0.3+\left(\frac{1-0.3}{4}\right)} = 2.105$

Speedup(infinity) = $\frac{1}{0.3} = 3.33$

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

# Amdahl’s Law

- **Example 2:** Suppose 25% of a sequential algorithm is the parallelizable portion. The remaining part must be calculated sequentially. Calculate the maximum theoretical speedup for the 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.

# Amdahl’s Law

- **Example 2:** Suppose 25% of a sequential algorithm is the parallelizable portion. The remaining part must be calculated sequentially. Calculate the maximum theoretical speedup for the parallel variant of this algorithm using 5 processors and infinite processors.

**Speedup(p=5) =** $\frac{1}{0.75+\left(\frac{1-0.75}{5}\right)} = 1.25$

**Speedup(p=5) =** $\frac{1}{0.75} = 1.333$

# 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 the 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.

*Proposed by Alan H. Karp and Horace P. Flatt in 1990. To determine to what extent an algorithm is parallelized. Here e is 0.75*

*One more use case: -*
*if metric value remains consistent w.r.t. increasing number of processors and speedups → You need to reduce sequential fraction.*
*if metric value increases as increasing number of processors and speedups → You need to parallelize overheads.*

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.

*Parallel Overhead: The amount of time required to coordinate **parallel** tasks, as opposed to doing useful work. **Parallel overhead** can include factors such as: Task start-up time. Synchronizations.*

# 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***

- Here, the 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 (or in this case, the same 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 cannot be parallelized
- The *output* of one stage is the *input* of the other stage
- Dividing the 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 the 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-CPUs with a shared memory

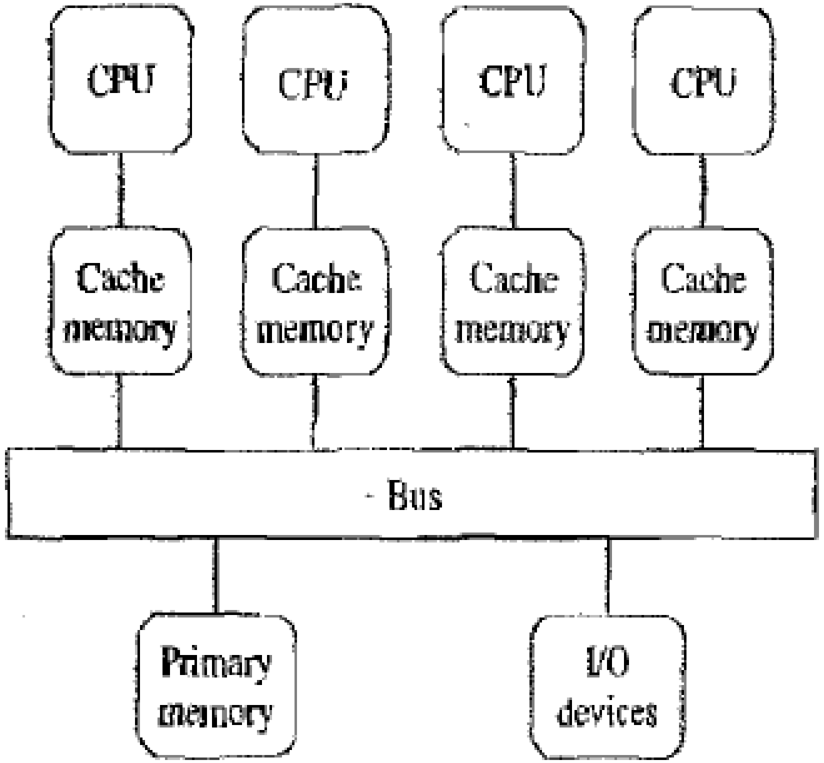
- The same address on two different CPUs refers to the same memory location.

- **Generally two categories:-**
  1. Centralized Multi-processors
  2. Distributed Multi-processor

# Multi-Processor

i. **Centralized Multi-processor**

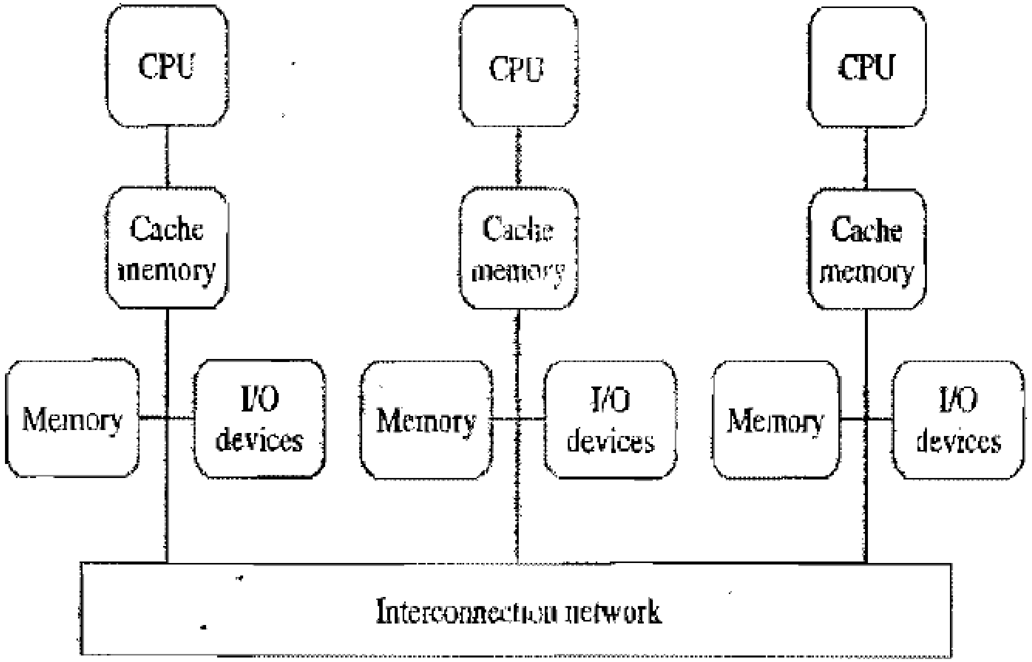
- Additional CPUs are attached to the system bus, and all the processors share the same primary memory
- All the memory is at one place and has the same access time from every processor
- Also known as **UMA** (Uniform Memory Access) multi-processor or **SMP** (symmetrical Multi-processor )



# 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



# Sources

- Slides of Dr. Rana Asif Rahman, FAST