

MULTIPLE PROCESSOR ORGANIZATION

- Single instruction, single data stream - SISD
- Single instruction, multiple data stream - SIMD
- Multiple instruction, single data stream - MISD
- Multiple instruction, multiple data stream - MIMD

SINGLE DATA STREAM - SISD

- Single processor
- Single instruction stream
- Data stored in single memory
- Uni-processor

MULTIPLE DATA STREAM - SIMD

- Single machine instruction
- Controls simultaneous execution
- Number of processing elements
- Lockstep basis
- Each processing element has associated data memory
- Each instruction executed on different set of data by different processors
- Vector and array processors

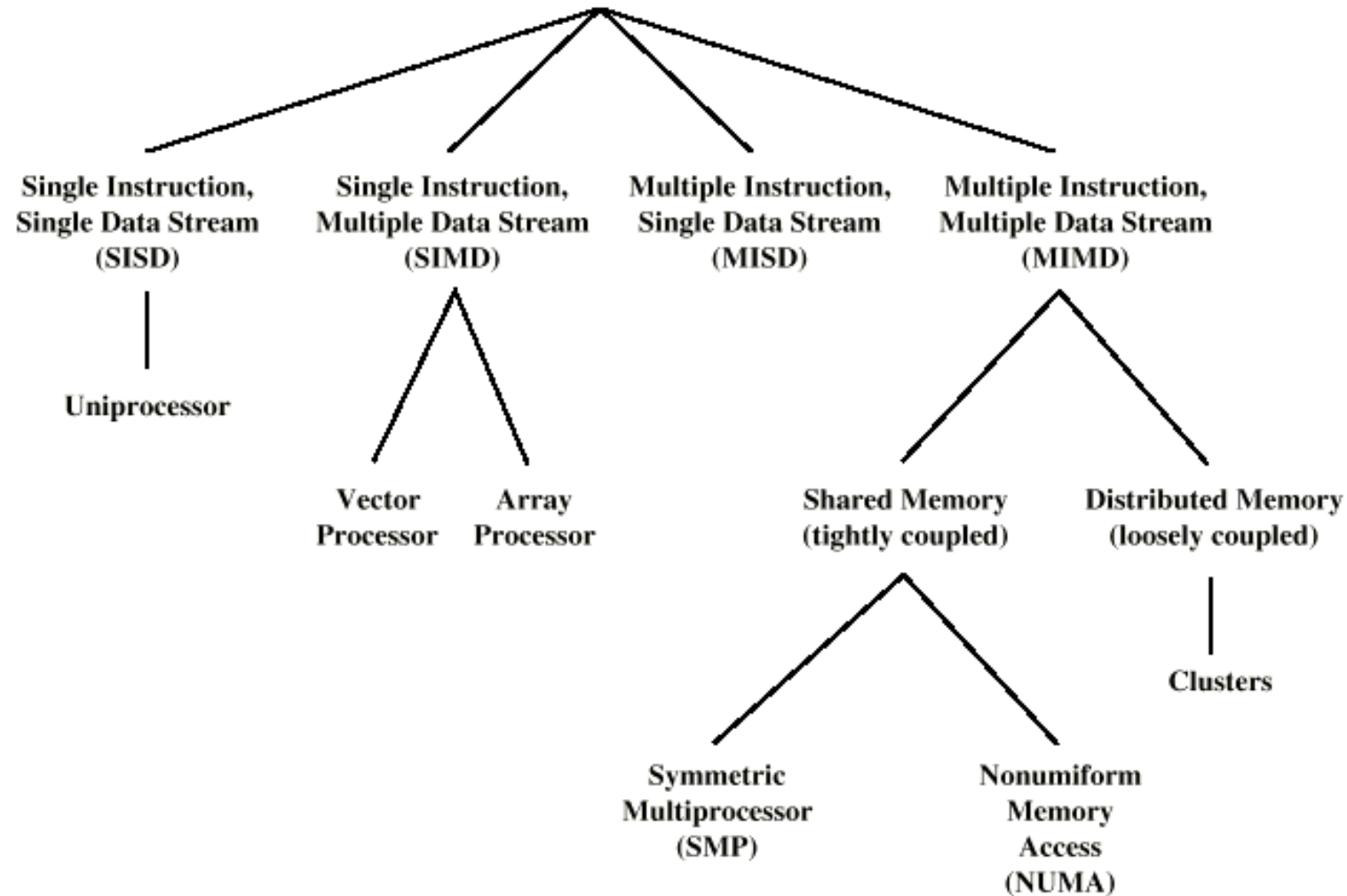
SINGLE DATA STREAM - MISD

- Sequence of data
- Transmitted to set of processors
- Each processor executes different instruction sequence
- Never been implemented

PROCESSOR

ARCHITECTURES

Processor Organizations



MIMD - OVERVIEW

- General purpose processors
- Each can process all instructions necessary
- Further classified by method of processor communication

TIGHTLY COUPLED - SMP

- Processors share memory
- Communicate via that shared memory
- Symmetric Multiprocessor (SMP)
- Share single memory or pool
- Shared bus to access memory
- Memory access time to given area of memory is approximately the same for each processor

TIGHTLY COUPLED - NUMA

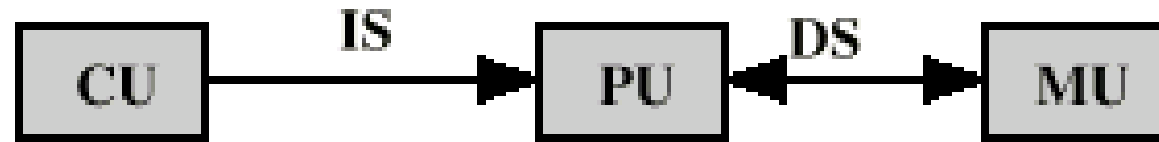
- Non-uniform memory access
- Access times to different regions of memory may differ.

LOOSELY COUPLED - CLUSTERS

- Collection of independent uniprocessors or SMPs
- Interconnected to form a cluster
- Communication via fixed path or network connections

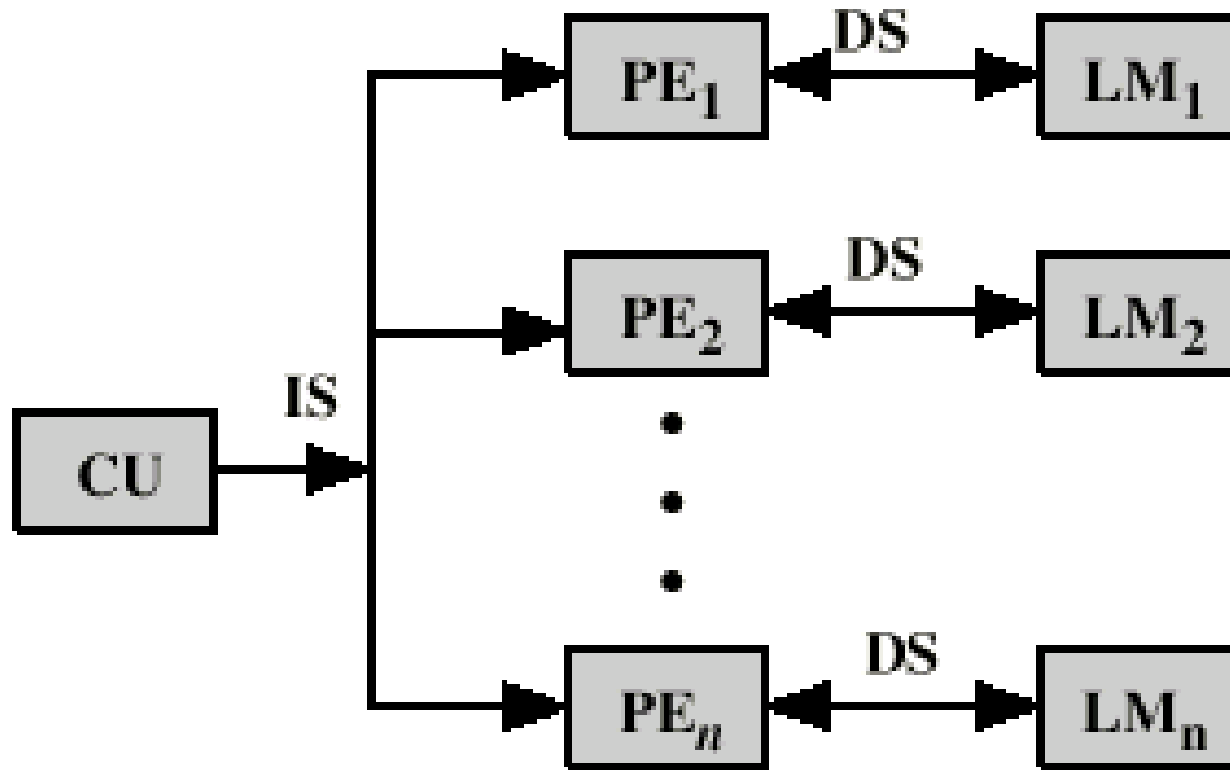
PARALLEL ORGANIZATIONS

- SISD



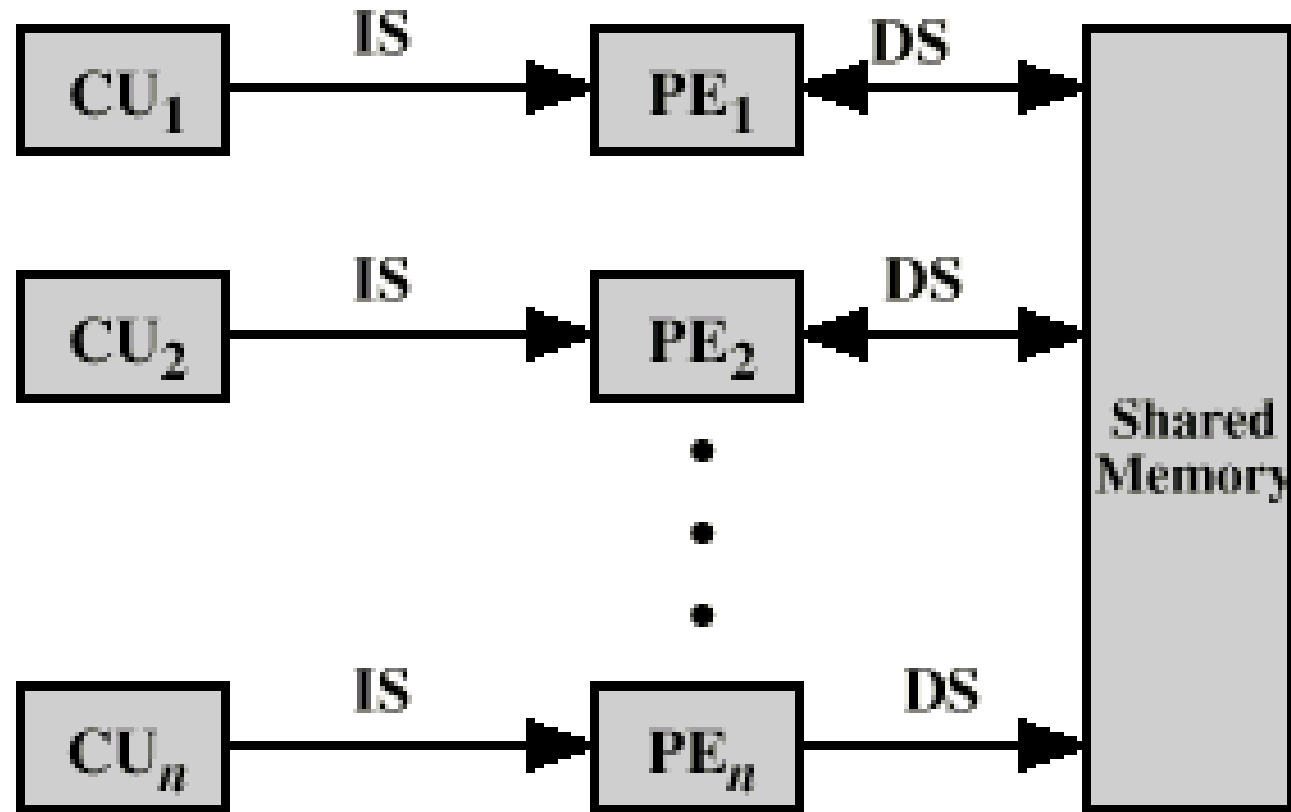
PARALLEL ORGANIZATIONS

- SIMD

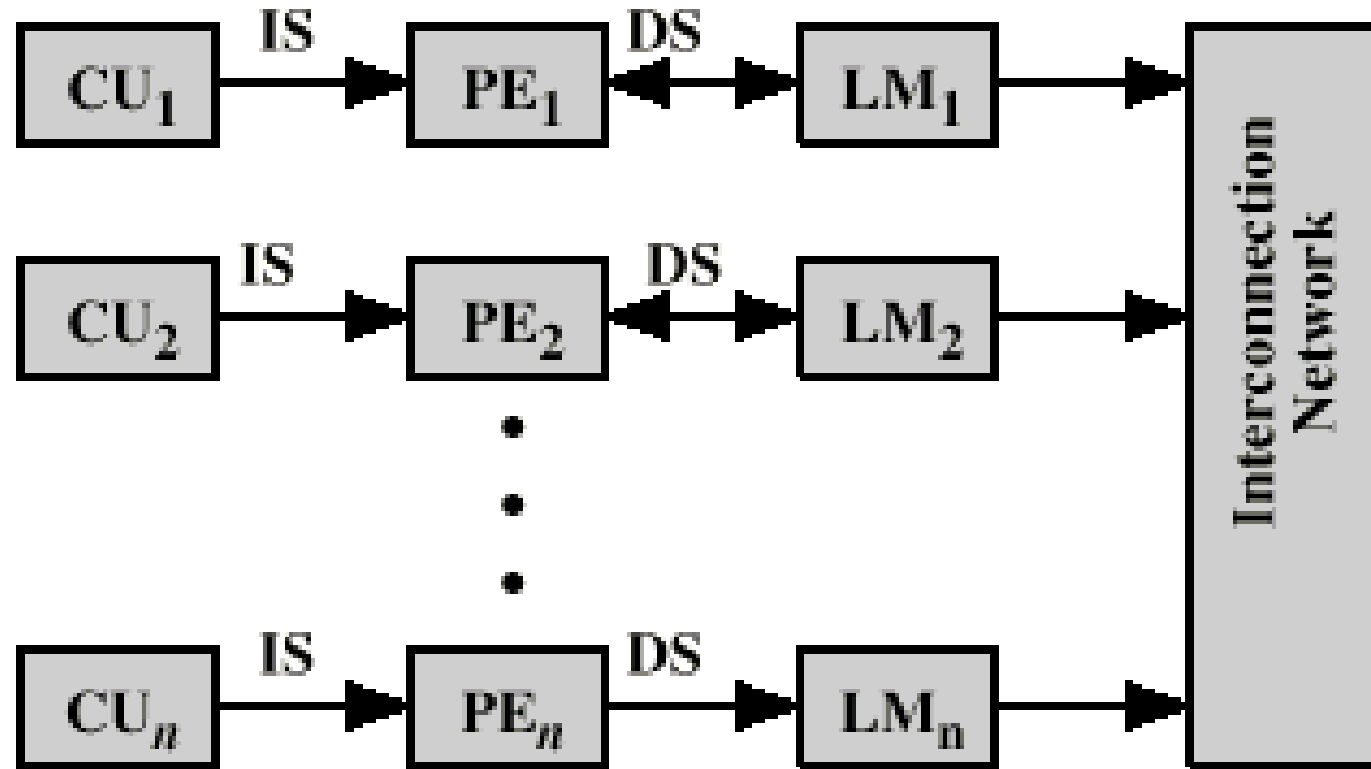


PARALLEL ORGANIZATIONS

- MIMD SHARED MEMORY



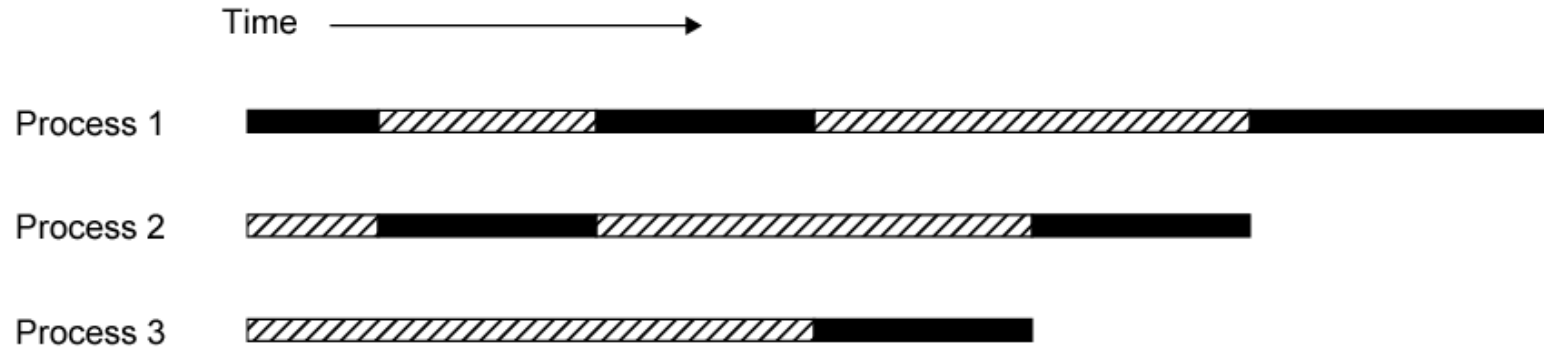
- MIMD DISTRIBUTED MEMORY



SYMMETRIC MULTIPROCESSORS

- A stand alone computer with the following characteristics
 - Two or more similar processors of comparable capacity
 - Processors share same memory and I/O
 - Processors are connected by a bus or other internal connection
 - Memory access time is approximately the same for each processor
 - All processors share access to I/O
 - Either through same channels or different channels giving paths to same devices
 - All processors can perform the same functions (hence symmetric)
 - System controlled by integrated operating system
 - providing interaction between processors
 - Interaction at job, task, file and data element

MULTIPROGRAMMING AND MULTIPROCESSING



(a) Interleaving (multiprogramming, one processor)



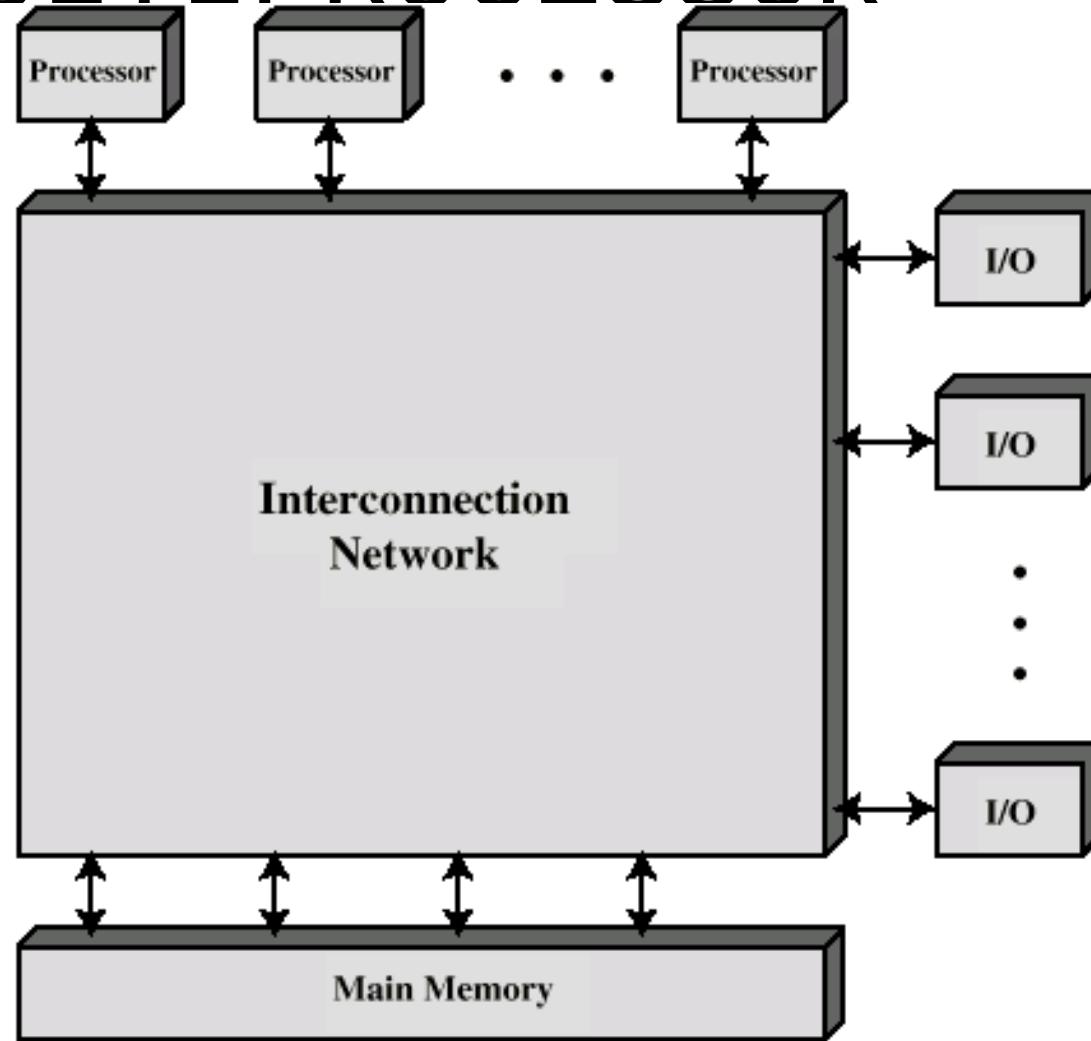
(b) Interleaving and overlapping (multiprocessing; multiple processors)

 Blocked  Running

SMP ADVANTAGES

- Performance
 - If some work can be done in parallel
- Availability
 - Since all processors can perform the same functions, failure of a single processor does not halt the system
- Incremental growth
 - User can enhance performance by adding additional processors
- Scaling
 - Vendors can offer range of products based on number of processors

TIGHTLY COUPLED MULTIPROCESSOR



MULTITHREADING AND CHIP MULTIPROCESSORS

- Instruction stream divided into smaller streams (threads)
- Executed in parallel
- Wide variety of multithreading designs

DEFINITIONS OF THREADS AND PROCESSES

- Thread in multithreaded processors may or may not be same as software threads
- Process:
 - An instance of program running on computer
 - Resource ownership
 - Virtual address space to hold process image
 - Scheduling/execution
 - Process switch

Cont...

- Thread: dispatch able unit of work within process
 - Includes processor context (which includes the program counter and stack pointer) and data area for stack
 - Thread executes sequentially
 - Interruptible: processor can turn to another thread
- Thread switch
 - Switching processor between threads within same process
 - Typically less costly than process switch

IMPLICIT AND EXPLICIT MULTITHREADING

- All commercial processors and most experimental ones use explicit multithreading
 - Concurrently execute instructions from different explicit threads
 - Interleave instructions from different threads on shared pipelines or parallel execution on parallel pipelines
- Implicit multithreading is concurrent execution of multiple threads extracted from single sequential program
 - Implicit threads defined statically by compiler or dynamically by hardware

APPROACHES TO EXPLICIT MULTITHREADING

- Interleaved
 - Fine-grained
 - Processor deals with two or more thread contexts at a time
 - Switching thread at each clock cycle
 - If thread is blocked it is skipped
- Blocked
 - Coarse-grained
 - Thread executed until event causes delay
 - E.g. Cache miss
 - Effective on in-order processor
 - Avoids pipeline stall

MULTIPROCESSOR SYSTEM

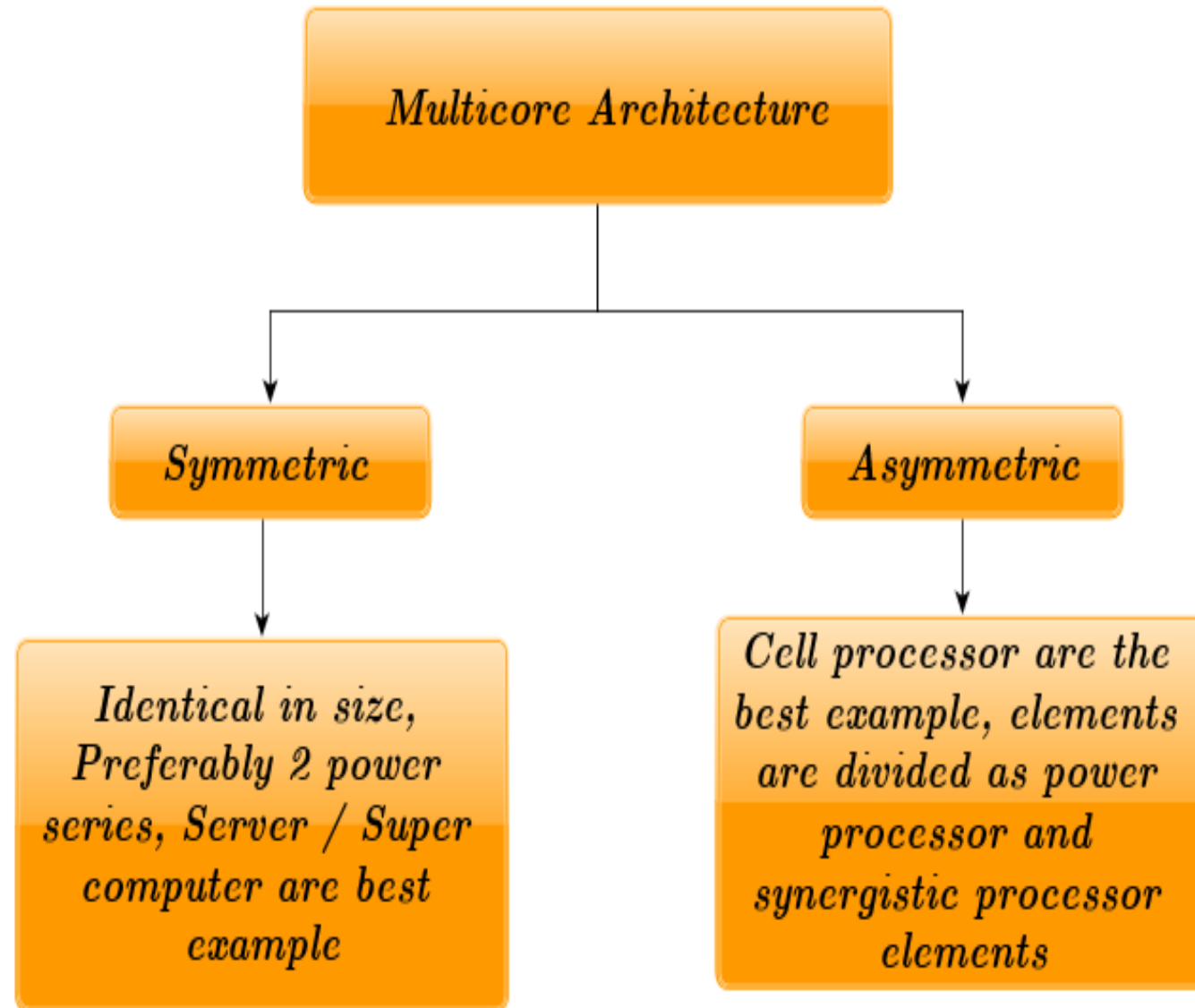
- A *multiprocessor system* is a single computer that includes multiple processors (computer modules).
- Processors may communicate and cooperate at different levels in solving a given problem.
- The communication may occur by sending messages from one processor to the other or by sharing a common memory.
- A *multiprocessor system* is controlled by one operating system which provides interaction between processors and their programs at the process, data set and data element levels.

MULTICOMPUTERS

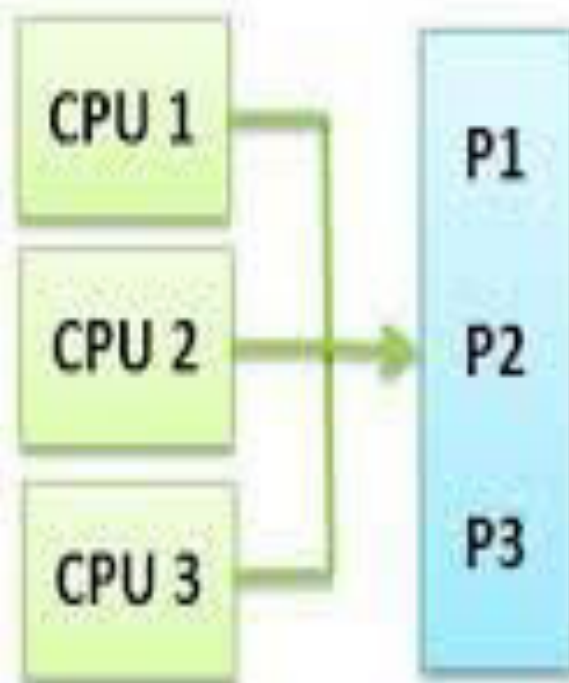
- There is a group of processors, in which each of the processors has sufficient amount of local memory.
- The communication between the processors is through messages.
- There is neither a common memory nor a common *clock*.
- This is also called distributed processing.

GRID COMPUTING

- Grid Computing enables geographically dispersed computers or computing clusters to dynamically and virtually share applications, data, and computational resources.
- It uses standard TCP/IP networks to provide transparent access to technical computing services wherever capacity is available, transforming technical computing into an information utility that is available across a department or organization.



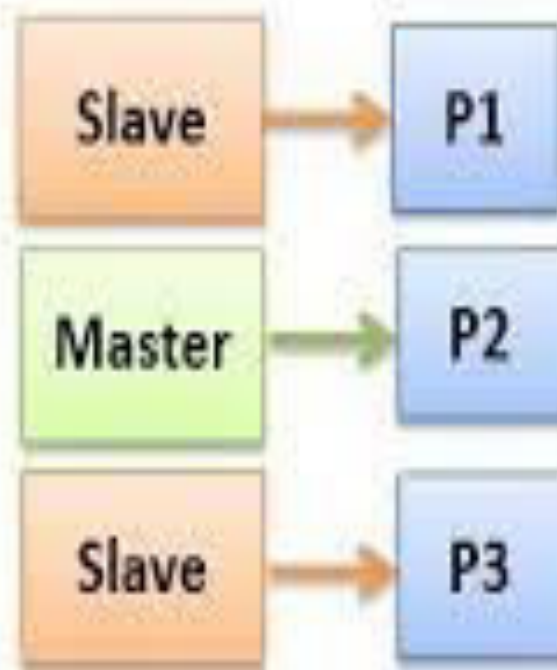
Symmetric Multiprocessing



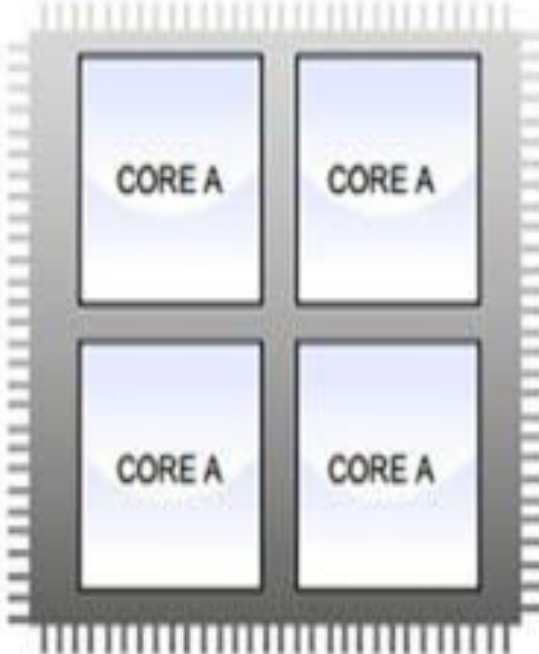
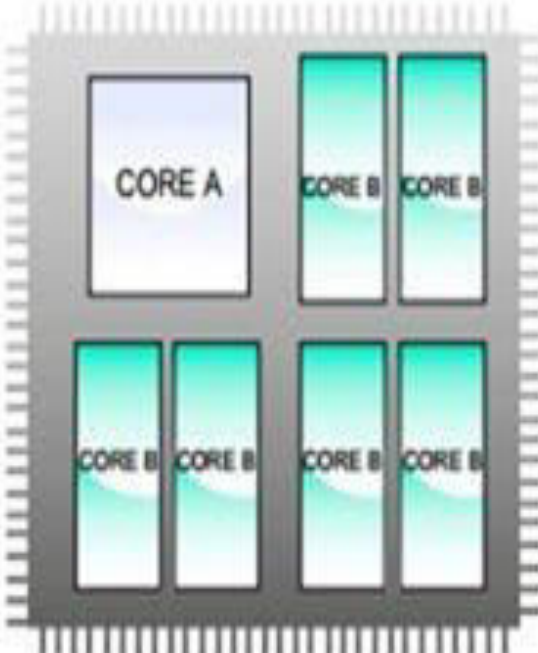
(Shared Memory)

Vs

Asymmetric Multiprocessing



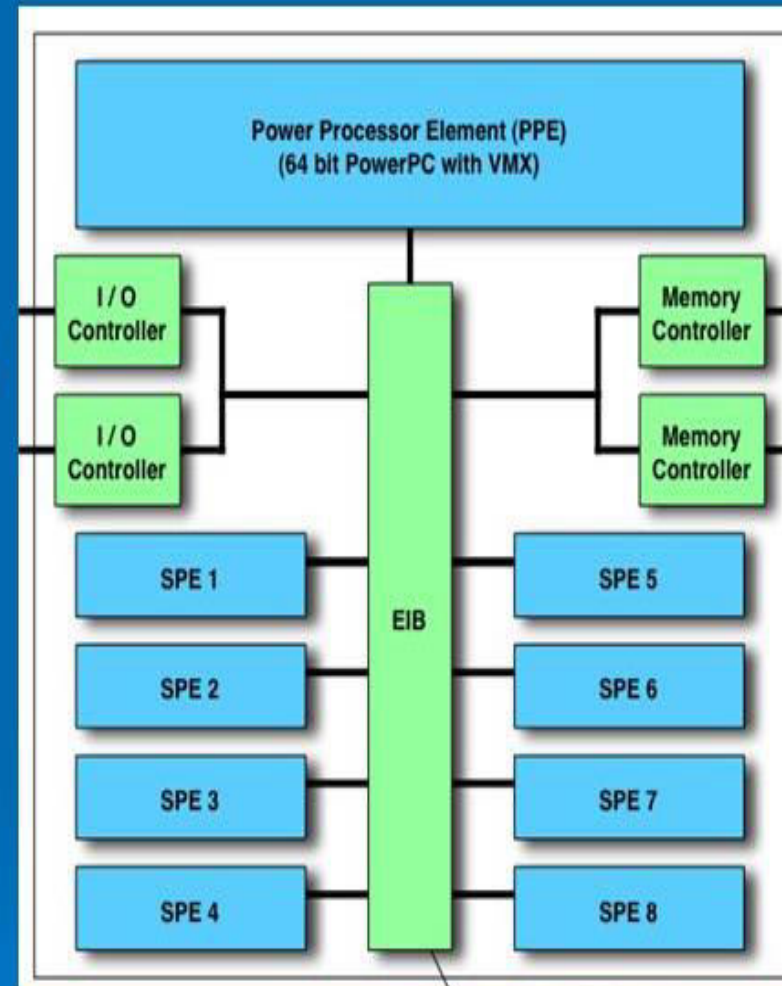
(No Shared Memory)

Homogeneous Multicore Processor Configuration	Heterogeneous Multicore Processor Configuration
	
Multiple cores of the same type are implemented in one CPU.	Multiple cores of different types are implemented in one CPU.

Multi-core processors

Asymmetric (heterogeneous) architecture

- IBM cell architecture: 9 cores
 - 1 PPE - power processor element
 - coordination and data transfer
 - 8 SPEs - Synergistic Processing Element
 - specialized mathematical units
 - applications:
 - supercomputers
 - playstations
 - home cinema
 - video cards



Challenges resulting from multi-core

- Relies on effective exploitation of multiple-thread parallelism
 - Need for parallel computing model and parallel programming model
- Aggravates memory wall
 - Memory bandwidth
 - Way to get data out of memory banks
 - Way to get data into multi-core processor array
 - Memory latency
 - Fragments L3 cache
- Pins become strangle point
 - Rate of pin growth projected to slow and flatten
 - Rate of bandwidth per pin (pair) projected to grow slowly
- Requires mechanisms for efficient inter-processor coordination
 - Synchronization
 - Mutual exclusion
 - Context switching

Advantages of Multi-core

- Cache coherency circuitry can operate at a much higher clock rate than is possible if the signals have to travel off-chip.
- Signals between different CPUs travel shorter distances, those signals degrade less.
- These higher quality signals allow more data to be sent in a given time period since individual signals can be shorter and do not need to be repeated as often.
- A dual-core processor uses slightly less power than two coupled single-core processors.

Performance

- ❑ Introduction
- ❑ Defining Performance
- ❑ The Iron Law of Processor Performance
- ❑ Processor performance enhancement
- ❑ Performance Evaluation Approaches
- ❑ Performance Reporting
- ❑ Amdahl's Law

Introduction

- ❑ Performance measurement is important:
 - ❑ Helps us to determine if one processor or computer works faster than other
 - ❑ Helps us to know how much performance improvement has taken place after incorporating some performance enhancement feature
 - ❑ Help to see through the marketing hype!
- ❑ Provides answer to the following questions:
 - ❑ Why is some hardware better than others for different programs?
 - ❑ What factors affect system

Defining Performance in terms of time

- ❑ Time is the final measure of computer performance

- ❑ A computer exhibits higher performance if it executes programs faster

Individual user concern

- ❑ Response Time (elapsed time, Latency):

 - ❑ How long does it take for my job to run?

 - ❑ How long does it take to execute (start to finish) my job?

 - ❑ How long must/wait for the database query?

System Manager concern

Execution Time

❑ Elapsed time

- ❑ Count everything (disk and memory access, waiting for IO, running other programs, etc) from start to finish

- ❑ A useful number, but often not good for comparison purpose

Elapsed time = CPU time + wait time (IO, other program, etc.)

❑ CPU time

- ❑ Doesn't count waiting for IO or time spent running other programs

- ❑ Can be divided into user CPU time and system CPU time (OS calls)

CPU time = user CPU time + System CPU Time

Elapsed time = user CPU time + System CPU Time + wait time

❑ Our focus: User CPU time

Measuring performance

- for some program running on machine X:

$$\text{Performance} = \frac{1}{\text{execution time}_x}$$

- X is n times faster than Y means:

$$\frac{\text{Performance}_x}{\text{Performance}_y} = n$$

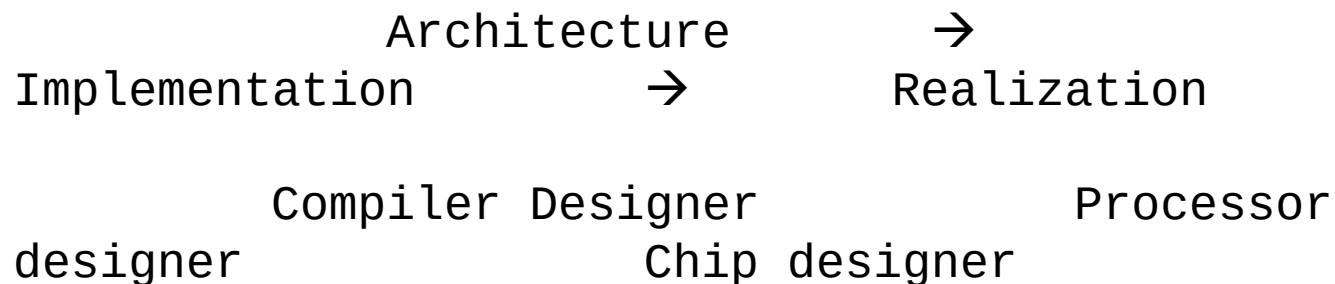
The IRON law of processor performance

$$\text{Processor performance} = \frac{\text{Time}}{\text{Program}}$$

$$= \frac{\text{Instruction}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Time}}{\text{Cycle}}$$

(code Size)
(Cycle time)

(CPI)



❑ **Instructions/Program (Instruction count)**

- Instructions executed, not static code size
- Determined by algorithm, compiler, ISA

❑ **Cycles/Instruction (CPI)**

- Determined by ISA and CPU organization
- Overlap among instructions reduces this term

❑ **Time/cycle (Cycle time)**

- Determined by technology, organization, clever circuit design

MIPS and MFLOPS

- ❑ Used extensively 30 years back.
- ❑ **MIPS**: millions of instructions processed per second.
- ❑ **MFLOPS**: Millions of Floating-point Operations completed per Second

$$\text{MIPS} = \frac{\text{Instruction Count}}{\text{Exec. Time} \times 10^6} = \frac{\text{Clock Rate}}{\text{CPI} \times 10^6}$$



Problems with MIPS

- ❑ Three significant problems with using MIPS:
- ❑ So severe, made some one term:
- ❑ “Meaningless Information about Processing Speed”
- ❑ Problem 1:
- ❑ MIPS is instruction set dependent.
- ❑ Problem 2:
- ❑ MIPS varies between programs on the same computer.
- ❑ Problem 3:
 - MIPS can vary inversely to performance!

❑ **Consider the following computer:**

The machine runs at 100MHz.

Instruction counts (in millions)
for each instruction class

Code type-	A (1 cycle)	B (2 cycle)	C (3 cycle)
Compiler 1	5	1	1
Compiler 2	10	1	1

Instruction A requires 1 clock cycle, Instruction B requires 2 clock cycles, Instruction C requires 3 clock cycles.

$$\text{CPI} = \frac{\text{CPU Clock Cycles}}{\text{Instruction Count}} = \frac{\sum_{i=1}^n \text{CPI}_i \times N_i}{\text{Instruction Count}}$$

$$CPI_1 = \frac{\overset{\text{count}}{[(5 \times 1) + (1 \times 2) + (1 \times 3)]} \overset{\text{cycles}}{\times 10^6}}{(5 + 1 + 1) \times 10^6} = 10/7 = 1.43$$

$$MIPS_1 = \frac{100 \text{ MHz}}{1.43} = 69.9$$

$$CPI_2 = \frac{[(10 \times 1) + (1 \times 2) + (1 \times 3)] \times 10^6}{(10 + 1 + 1) \times 10^6} = 15/12 = 1.25$$

$$MIPS_2 = \frac{100 \text{ MHz}}{1.25} = 80.0$$

So, compiler 2 has a higher MIPS rating and should be faster?

❑ Now let's compare CPU time:

$$\text{CPU Time} = \frac{\text{Instruction Count} \times \text{CPI}}{\text{Clock Rate}}$$

$$\text{CPU Time}_1 = \frac{7 \times 10^6 \times 1.43}{100 \times 10^6} = 0.10 \text{ seconds}$$

$$\text{CPU Time}_2 = \frac{12 \times 10^6 \times 1.25}{100 \times 10^6} = 0.15 \text{ seconds}$$

Therefore program 1 is faster despite a lower MIPS!

Computer Performance

“X is N% faster than Y.”

$$\frac{\text{Execution Time of Y}}{\text{Execution Time of X}} = 1 + \frac{N}{100}$$

Amdahl's law for overall speedup

$$\text{Overall Speedup} = \frac{1}{(1 - F) + \frac{F}{S}}$$

F = The fraction enhanced

S = The speedup of the enhanced fraction

Using Amdahl's law

Overall speedup if we make 90% of a program run 10 times faster.

$$F = 0.9 \quad S = 10$$

$$\text{Overall Speedup} = \frac{1}{(1 - 0.9) + \frac{0.9}{10}} = \frac{1}{0.1 + 0.09} = 5.26$$

Overall speedup if we make 80% of a program run 20% faster.

$$F = 0.8 \quad S = 1.2$$

$$\text{Overall Speedup} = \frac{1}{(1 - 0.8) + \frac{0.8}{1.2}} = \frac{1}{0.2 + 0.66} = 1.153$$

Amdahl's Law

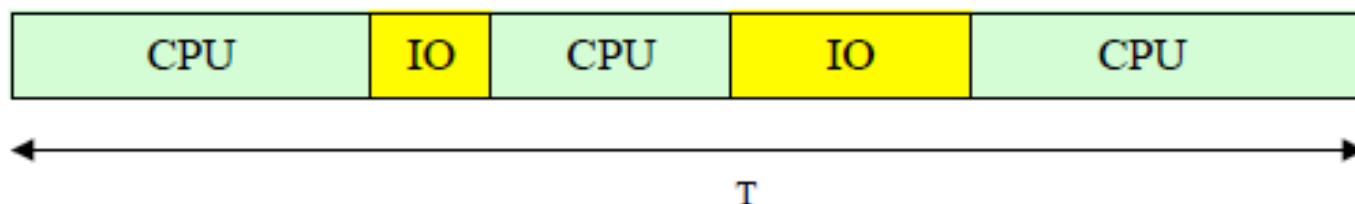
How is **system performance** altered when some **component is changed**?

Example 1:

Program execution time is made up of 75% CPU time and 25% I/O time. Which is the better enhancement:

(a) Increasing the CPU speed by 50% or (b) reducing I/O time by half?

Execution model: No overlap between CPU and I/O operations



Program execution time $T = T_{\text{cpu}} + T_{\text{io}}$

$$T_{\text{cpu}} / T = 0.75 \text{ and } T_{\text{io}} / T = 0.25$$

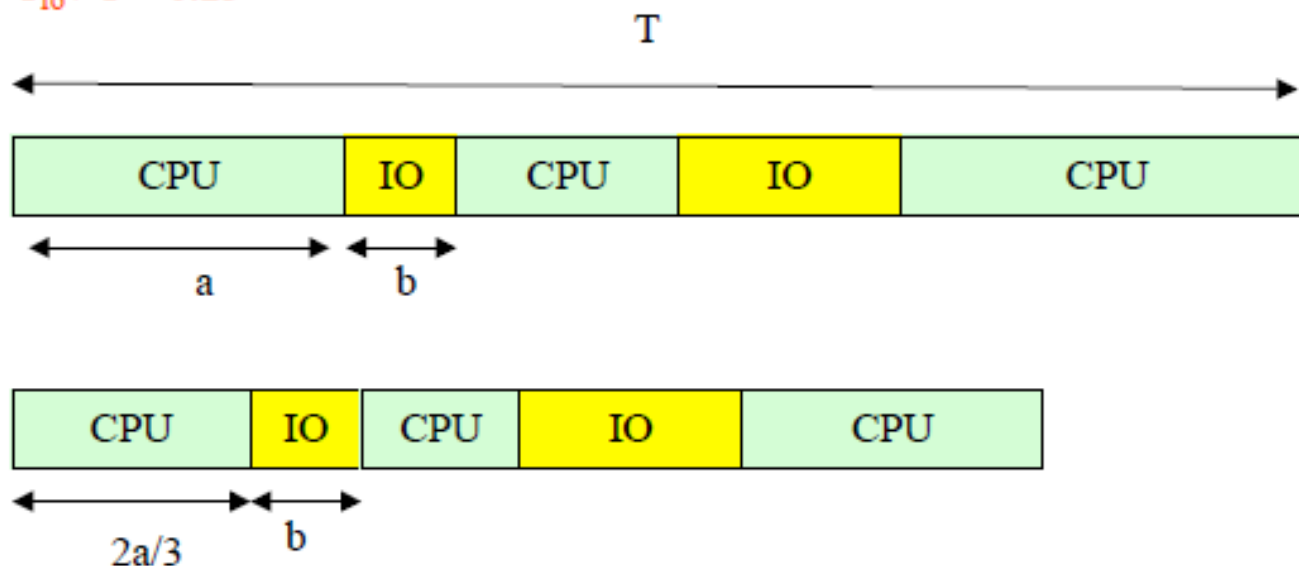
Amdahl's Law

(a) Increasing the CPU speed by 50%

Program execution time $T = T_{\text{cpu}} + T_{\text{io}}$ $T_{\text{old}} = T$

$$T_{\text{cpu}} / T = 0.75$$

$$T_{\text{io}} / T = 0.25$$



Program execution time $T_{\text{new}} = T_{\text{cpu}} / 1.5 + T_{\text{io}}$

$$T_{\text{new}} = T_{\text{cpu}} / 1.5 + T_{\text{io}} = 0.75 T / 1.5 + 0.25T = 0.75T$$

For a 50% improvement in CPU speed: Execution time decreases by 25%

$$\text{Speedup} = T_{\text{old}} / T_{\text{new}} = T / 0.75T = 1.33$$

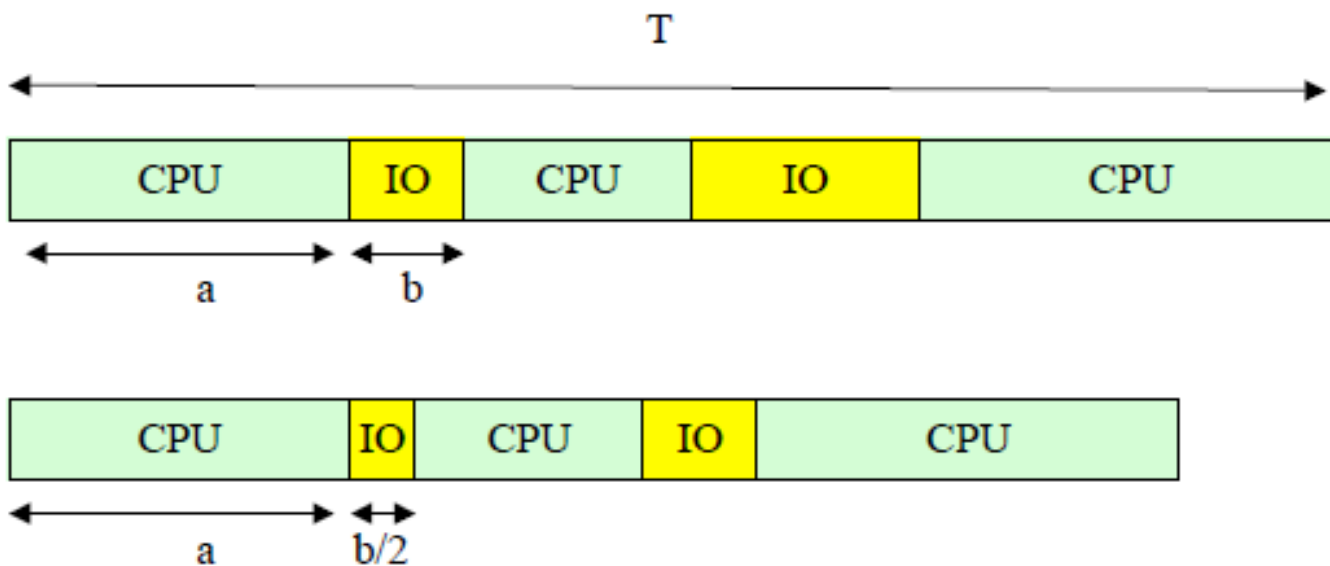
Amdahl's Law

(b) Halve the IO Time

Program execution time $T = T_{\text{cpu}} + T_{\text{io}}$ $T_{\text{old}} = T$

$$T_{\text{cpu}} / T = 0.75$$

$$T_{\text{io}} / T = 0.25$$



Program execution time $T_{\text{new}} = T_{\text{cpu}} + T_{\text{io}} / 2$

$$T_{\text{new}} = 0.75 T + 0.25 T / 2 = 0.875 T$$

For a 100% improvement in IO speed: Execution time decreases by 12.5%

$$\text{Speedup} = T_{\text{old}} / T_{\text{new}} = T / 0.875 T = 1.14$$

Amdahl's Law

Limiting Cases

- CPU speed improved infinitely so T_{CPU} tends to zero
 $T_{\text{new}} = T_{\text{IO}} = 0.25T$ Speedup limited to 4
- IO speed improved infinitely so T_{IO} tends to zero
 $T_{\text{new}} = T_{\text{CPU}} = 0.75T$ Speedup limited to 1.33

Problem

- Improving performance
 - Current system
 - * Execution time – 10 sec
 - * Clock speed – 400 MHz
 - New system
 - * Execution time – 6 sec
 - * Clock speed – ?
 - * Number of clock cycles – 1.2 times current system
 - Compute the number of clock cycles for current system
 - *

$$\begin{aligned}\text{CPU time} &= \frac{\text{CPU clock cycles for program}}{\text{Clock rate}} \\ 10\text{sec} &= \frac{\text{CPU clock cycles for program}}{400 \times 10^6 \text{cps}}\end{aligned}$$

$$* \text{ CPU clock cycles for program} = 4000 \times 10^6$$

- Compute the clock speed for new system

*

$$\text{CPU time} = \frac{\text{CPU clock cycles for program}}{\text{Clock rate}}$$

$$6sec = \frac{1.2 \times 4000 \times 10^6}{\text{Clock rate}}$$

*

$$\begin{aligned}\text{Clock rate} &= \frac{1.2 \times 4000 \times 10^6}{6} \\ &= 800 \times 10^6 \\ &= 800\text{MHz}\end{aligned}$$

- Basic performance equation

$$\text{CPU time} = \text{Instruction count} \times \text{CPI} \times \text{Clock cycle time}$$

or

$$\text{CPU time} = \frac{\text{Instruction count} \times \text{CPI}}{\text{Clock rate}}$$

$$\text{CPU clock cycles} = \sum_{i=1}^n (\text{CPI}_i \times C_i)$$

- * C_i is the number of instructions of class i
- * CPI_i is the average number of cycles per instruction for class i
- * n is the number of instruction classes

Code sequence	Number of instructions		
	A	B	C
c_1	2	1	2
c_2	4	1	1

Find out the number of instructions for each code sequence, the faster code sequence, and CPI for each code sequence

Number of instructions in sequence $c_1 = 2 + 1 + 2 = 5$

Number of instructions in sequence $c_2 = 4 + 1 + 1 = 6$

Obviously, sequence c_1 executes fewer instructions

$$\text{CPU clock cycles}_1 = (2 \times 1) + (1 \times 2) + (2 \times 3) = 2 + 2 + 6 = 10$$

$$\text{CPU clock cycles}_2 = (4 \times 1) + (1 \times 2) + (1 \times 3) = 4 + 2 + 3 = 9$$

Code sequence c_2 is faster

$$\text{CPI} = \frac{\text{CPU clock cycles}}{\text{Instruction count}}$$

$$\text{CPI}_1 = \frac{10}{5} = 2$$

$$\text{CPI}_2 = \frac{9}{6} = 1.5$$

Bench mark sample with problem

- A program runs in 100 sec on a machine, with multiply operations taking up 80 seconds of this time. How much does the speed of multiplication need to improve to get a five-fold increase in code execution?

$$\text{Execution time after improvement} = \frac{\text{Execution time affected by improvement}}{\text{Amount of improvement}} + \text{Execution time unaffected}$$

$$\frac{100}{5} = \frac{80}{n} + (100 - 80)$$

$$20 = \frac{80}{n} + 20$$

$$0 = \frac{80}{n}$$

There is no amount by which we can improve the performance of multiply to realize a five-fold increase in overall performance

- This is *Amdahl's Law* in computing, or the law of diminishing returns in everyday life
- Opportunity of improvement is affected by how many time the event occurs