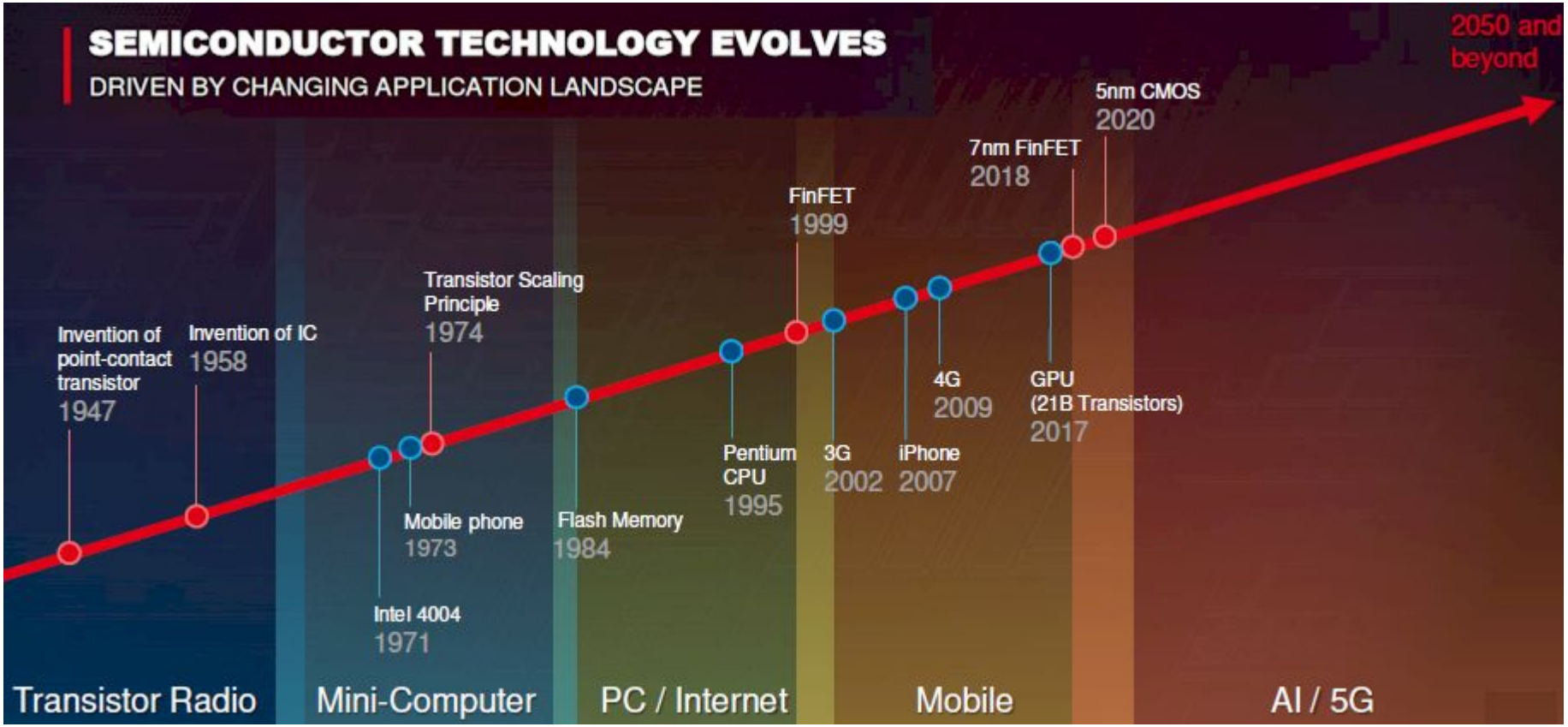
# Module 2




# Performance Evaluation of Computer Systems

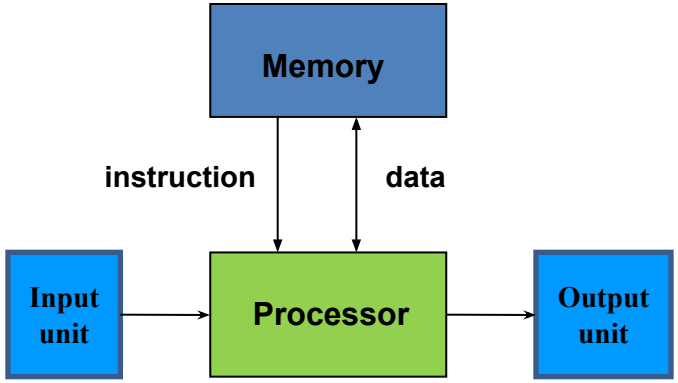
# Content

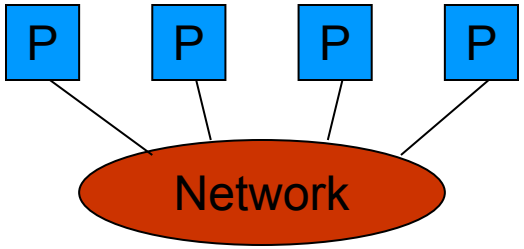
- Evaluation Parameters
- Quantitative Principals: Amdahl's Law

# A Computer System

**Computer systems** consist of:

- Processor
- Memory
- Input/Output
- Operating system
- Network



# Performance Factors

**Performance** depends on:

- Technology
- Instruction Set Architecture
- Organization
- Software

# Technology

- In recent years, microprocessors have become **smaller** and **denser**.

# Technology

| | 1945 | 2020 |
|---|---|---|
| Computer | ENIAC | Laptop |
| Devices | 18 000 | 17 000 000 000 |
| Weight (kg) | 27 200 | 1.5 |
| Size ($m^3$) | 68 | 0.0018 |
| Power (watts) | 20 000 | 5.5 |
| Cost ($) | 4 630 000 | 1 000 |
| Memory (bytes) | 200 | 160 000 000 000 |
| Performance (Flops/s) | 800 | 3 000 000 000 |

# Performance Units

**Speed**

| | | |
|---|---|---|
| 1 Mflop/s | 1 Megaflop/s | $10^6$ Flop/second |
| 1 Gflop/s | 1 Gigaflop/s | $10^9$ Flop/second |
| 1 Tflop/s | 1 Teraflop/s | $10^{12}$ Flop/second |
| 1 Pflop/s | 1 Petaflop/s | $10^{15}$ Flop/second |
| 1 Eflop/s | 1 Exaflop/s | $10^{18}$ Flop/second |

**Storage**

| | | |
|---|---|---|
| 1 MB | 1 Megabyte | $10^6$ Bytes |
| 1 GB | 1 Gigabyte | $10^9$ Bytes |
| 1 TB | 1 Terabyte | $10^{12}$ Bytes |
| 1 PB | 1 Petabyte | $10^{15}$ Bytes |

# Performance Evaluation of Computer Systems

Performance depends on:

- Technology
- Instruction Set Architecture
- Organization
- Software

# Instruction Set Architecture-ISA

Instruction Set Design:

- RISC / CISC
  - Code density
- Number of operands
  - Stack machines (0-operand)
  - Accumulator machines (1-operand)
  - Register machines (2-operand, 3-operand)

# RISC VS CISC

| CISC | RISC |
|---|---|
| CISC architecture gives more importance to hardware | RISC architecture gives more importance to software |
| Multiple Instruction sizes and formats | Instruction of same size with few formats |
| Less registers | Uses more registers |
| More addressing modes | Fewer addressing modes |
| Extensive use of microprogramming | Complexity in compiler |
| Instructions take varying amount of cycle time | Instructions take one cycle time |
| Pipelining is difficult | Pipelining is easy |

# Performance Evaluation of Computer Systems

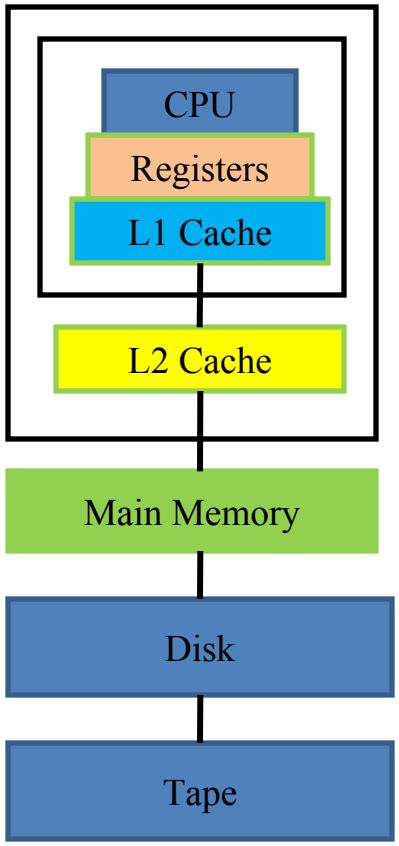
Performance depends on:

- Technology
- Instruction Set Architecture
- Organization
- Software

# Organization

## Memory Hierarchy

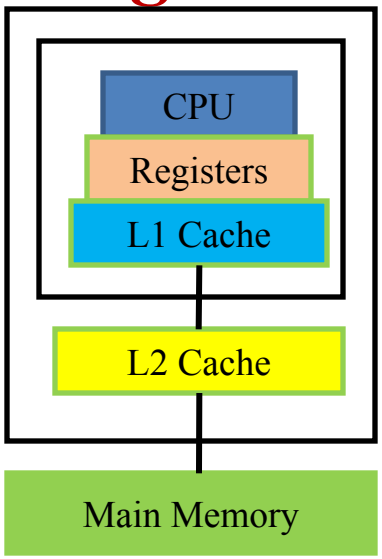
| Hierarchy | Speed | Size |
|---|---|---|
| Within the processor (CPU-registers-on chip cache) | 1 ns | Byte |
| L2 cache (SRAM) | 10 ns | KByte |
| Main Memory (DRAM) | 100 ns | MByte |
| Secondary storage (Disk) | 10 ms | Gbyte |
| Tertiary Storage (Tape/Disk) | 10 s | TByte |



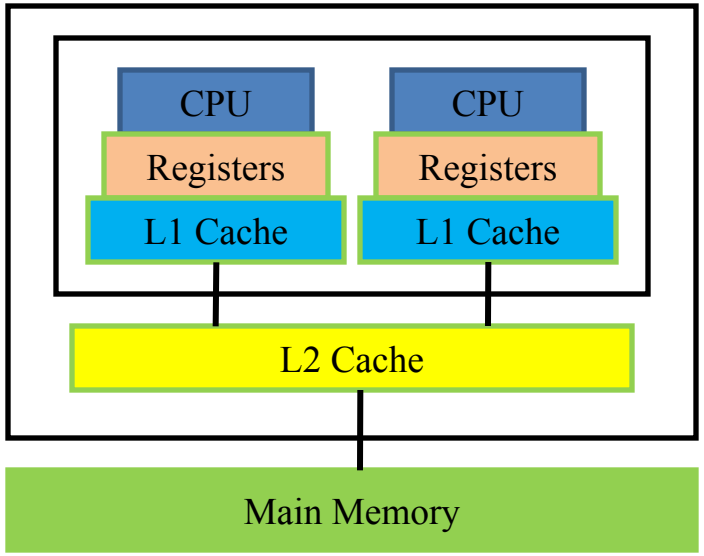
# Organization

## Multicore Chips

***Single-core***




***Dual-core***



# Performance Evaluation of Computer Systems

Performance depends on:

- Technology
- Instruction Set Architecture
- Organization
- Software

Performance Evaluation of Computer Systems

# EVALUATION PARAMETERS [3]

# The Questions are..

- How to compare performance of two different machines?
- What factors affect performance?
- How to improve performance?

## Defining Performance [ 4]

- If you were running a program on two different desktop computers, you'd say that the faster one is the desktop computer that gets the job done first.

- If you were running a datacenter that had several servers running jobs submitted by many users, you'd say that the faster computer was the one that completed the most jobs during a day.

- As an individual computer user, you are interested in reducing response time—the time between the start and completion of a task—also referred to as execution time.

- Datacenter managers are often interested in increasing throughput or bandwidth—the total amount of work done in a given time.

- Hence, in most cases, we will need different performance metrics

- **Response Time**
  - the time between the start and completion of a task also referred to as **execution time.**
- **Throughput**
  - The total amount of work done in a given time.

- **Decreasing response time almost always improves throughput.**

- To maximize performance, we want to minimize response time or execution time for some task. Thus, we can relate performance and execution time for a computer X:

- $\text{Performance}_X = 1/\text{Execution time}_X$

- This means that for two computers X and Y, if the performance of X is greater than the performance of Y, we have

- $\text{Performance}_x > \text{Performance}_y$

- $1/\text{Execution time}_x > 1/\text{Execution time}_y$

- $\text{Execution time}_Y > \text{Execution time}_X$
- That is, the execution time on Y is longer than that on X, if X is faster than Y.

- we oft en want to relate the performance of two different computers quantitatively. We will use the phrase “X is *n times faster than* Y”—or equivalently “X is *n times as fast as Y”—to mean*

- $\text{Performance}_x / \text{Performance}_y = n$

- If X is *n times as fast as Y, then the execution time on Y is n times as long as it is*
- on X:

- $\text{Performance}_x / \text{Performance}_y = \text{Execution time}_y / \text{Execution time}_x = n$ = *Speedup*

- If computer A runs a program in 10 seconds and computer B runs the same program in 15 seconds, how much faster is A than B?
- We know that A is *n times as fast as B if*

- $\text{Performance}_A / \text{Performance}_B = \text{Execution time}_B / \text{Execution time}_A = n$

- Thus the performance ratio is

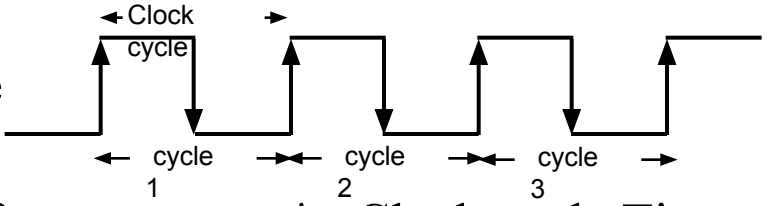
- $15 / 10 = 1.5$

- A is therefore 1.5 times as fast as B.
- In the above example, we could also say that computer B is 1.5 times *slower than*
- computer A, since
- $\text{Performance}_A / \text{Performance}_B = 1.5$

- $\text{Performance}_A = 1.5 * \text{Performance}_B$

# CPU Performance Its Factors[4]

Execution time depends on Clock Cycle time




CPU execution time = CPU clock cycles for a program * Clock cycle Time

- Clock Cycle time= 1 / Clock rate,
- So,

CPU execution time for a program = CPU clock cycles for a program /Clock rate

Example :

Our favorite program runs in 10 seconds on computer A, which has a 2 GHz clock. We are trying to help a computer designer build a computer, B, which will run this program in 6 seconds. The designer has determined that a substantial increase in the clock rate is possible, but this increase will affect the rest of the CPU design, causing computer B to require 1.2 times as many clock cycles as computer A for this program. What clock rate should we tell the designer to target?

# Solution

- Clock cycle$_A$ = Execution Time$_A$ * Clock Rate
- = 10 sec* 2*10^9 cycle/ sec =20*10^9 cycle
- Clock Cycle$_B$= 1.2 * Clock cycle$_A$
- = 24 * 10^9 cycle
- Clock Rate$_B$ = Clock Cycle$_B$ / Execution time$_B$
- = 24 * 10^9 cycle / 6 sec
- = 4 *10^9 cycle / sec = 4GHz

The CPU clock cycle required for a program can be define as,

**CPU clock cycles for a program= Instructions for a program * Average clock cycles per instruction**

**Average Clock cycle per Instruction-**

Defined as average number of clock taken by each instruction. Which is also known as CPI.

- Different instructions may take different times for execution , the CPI is the average of all instructions executed in the program.

## Example

- Suppose we have two implementations of the same instruction set architecture. Computer A has a clock cycle time of 250ps and a CPI of 2.0 for some program, and computer B has a clock cycle time of 500ps and a CPI of 1.2 for the same program. Which computer is faster for this program and by how much?.

- CPU clock cycles$_A$ = I* 2.0
- CPU clock cycle$_B$ = I* 1.2
- CUP time$_A$ = CPU clock cycle$_A$ * Clock cycle time
- = I * 2.0 * 250 ps = 500 * I ps
- CPU time $_B$ = I * 1.2 * 500 ps = 600 * I ps
- CPU performance$_A$ / CPU performance$_B$ = Execution time$_B$ / Execution time$_A$
- = 600 *I ps / 500* I ps =1.2
- So A is 1.2 times as fast as B .

Now we can write CPU execution time as,

- **CPU time = Instruction count * CPI * Clock cycle time**

- **CPU time = ( Instruction count * CPI ) / Clock rate**

# CPU Execution Time: Example

- A Program is running on a specific machine (CPU) with the following parameters:
  - Total executed instruction count: 10,000,000 instructions
  - Average CPI for the program: 2.5 cycles/instruction.
  - CPU clock rate: 200 MHz.

What is the execution time for this program??

CPU time = Instruction count x CPI x Clock cycle Time

= 10,000,000 x 2.5 x 1 / clock rate

= 10,000,000 x 2.5 x $5 \times 10^{-9}$

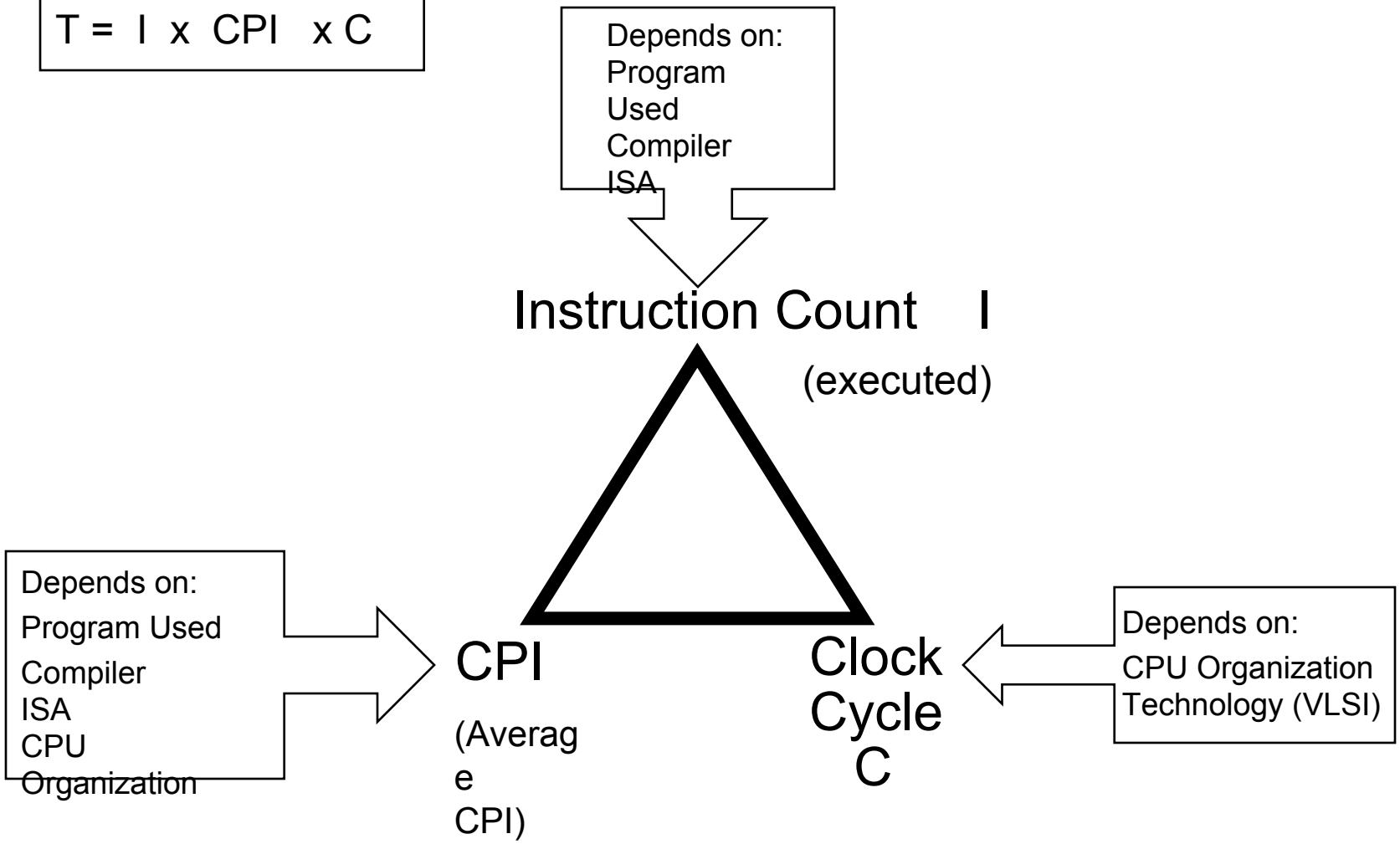
= 0.125 seconds

$T = I \times CPI \times C$

# Aspects of CPU Execution Time

CPU Time = Instruction count executed x CPI x Clock cycle

T = I x CPI x C



# Factors Affecting CPU Performance

| T = I x CPI x C | Instruction Count | Cycles per Instruction | Clock Rate (1/C) |
|---|---|---|---|
| Program | | | |
| Compiler | | | |
| Instruction Set Architecture (ISA) | | | |
| Organization (CPU Design) | | | |
| Technology (VLSI) | | | |

$$T = I \times CPI \times C$$

# Performance Comparison: Example

A Program is running on a specific machine (CPU) with the following parameters:

- Total executed instruction count, I: 10,000,000 instructions
- Average CPI for the program: 2.5 cycles/instruction.
- CPU clock rate: 200 MHz.

Thus: C = $1/(200 \times 10^6) = 5 \times 10^{-9}$ seconds

- Using the same program with these changes:
  - A new compiler used: New executed instruction count, I: 9,500,000
    New CPI: 3.0
  - Faster CPU implementation: New clock rate = 300 MHz
- What is the speedup with the changes?

$$\text{Speedup} = \frac{\text{Old Execution Time}}{\text{New Execution Time}} = \frac{I_{old} \times CPI_{old} \times \text{Clock cycle}_{old}}{I_{new} \times CPI_{new} \times \text{Clock Cycle}_{new}}$$

Speedup = $(10{,}000{,}000 \times 2.5 \times 5\times10^{-9}) / (9{,}500{,}000 \times 3 \times 3.33\times10^{-9})$

= .125 / .095 = 1.32

or 32 % faster after changes.

Clock Cycle = C = 1/ Clock Rate

$T = I \times CPI \times C$

# Instruction Types & CPI

- Given a program with $n$ types or classes of instructions executed on a given CPU with the following characteristics:

$C_i$ = Count of instructions of type$_i$

$CPI_i$ = Cycles per instruction for type$_i$ i = 1, 2, …. n

Depends on CPU Design

Then:

CPI = CPU Clock Cycles / Instruction Count I

i.e average or effective CPI

Executed

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

Executed Instruction Count I = $\Sigma\ C_i$

T = I x CPI x C

# Instruction Types & CPI: An Example

- An instruction set has three instruction classes:

| Instruction class | CPI |
|---|---|
| A | 1 |
| B | 2 |
| C | 3 |

For a specific CPU design

- Two code sequences have the following instruction counts:

| | Instruction counts for instruction class | | |
|---|---|---|---|
| Code Sequence | A | B | C |
| 1 | 2 | 1 | 2 |
| 2 | 4 | 1 | 1 |

- CPU cycles for sequence 1 = 2 x 1 + 1 x 2 + 2 x 3 = 10 cycles
  CPI for sequence 1 = clock cycles / instruction count
  = 10 /5 = 2

i.e average or effective CPI

- CPU cycles for sequence 2 = 4 x 1 + 1 x 2 + 1 x 3 = 9 cycles
  CPI for sequence 2 = 9 / 6 = 1.5

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

CPI = CPU Cycles /

# Instruction Frequency & CPI

- Given a program with $n$ types or classes of instructions with the following characteristics:
- $C_i$ = Count of instructions of type$_i$ executed

i = 1, 2, …. n

- $CPI_i$ = Average cycles per instruction of type$_i$
- $F_i$ = Frequency or fraction of instruction type$_i$ executed
- = $C_i$/ total executed instruction count = $C_i$/ I
- Then:

Where: Executed Instruction Count I = $\Sigma C_i$

$$CPI = \sum_{i=1}^{n} (CPI_i \times F_i)$$

i.e average or effective CPI

Fraction of total execution time for instructions of type i = $\frac{CPI_i \times F_i}{CPI}$

T = I x CPI x C

# Instruction Type Frequency & CPI: A RISC Example

$\frac{CPI_i \times F_i}{CPI}$

Program Profile or Executed Instructions Mix

Base Machine (Reg / Reg)

Depends on CPU Design

| Op | Freq, $F_i$ | $CPI_i$ | $CPI_i \times F_i$ | % Time |
|---|---|---|---|---|
| ALU | 50% | 1 | .5 | 23% = .5/2.2 |
| Load | 20% | 5 | 1.0 | 45% = 1/2.2 |
| Store | 10% | 3 | .3 | 14% = .3/2.2 |
| Branch | 20% | 2 | .4 | 18% = .4/2.2 |
| | | | Sum = 2.2 | |

Given

Typical Mix

$$CPI = \sum_{i=1}^{n} (CPI_i \times F_i)$$

i.e average or effective CPI

CPI = .5 x 1 + .2 x 5 + .1 x 3 + .2 x 2 = 2.2

= 5 + 1 + .3 + .4

T = I x CPI x C

# Computer Performance Measures : MIPS (Million Instructions Per Second) Rating

- For a specific program running on a specific CPU the MIPS rating is a measure of how many millions of instructions are executed per second:

$$\begin{aligned}\text{MIPS Rating} &= \text{Instruction count} \;/\; (\text{Execution Time} \times 10^6)\\ &= \text{Instruction count} \;/\; (\text{CPU clocks} \times \text{Cycle time} \times 10^6)\\ &= (\text{Instruction count} \times \text{Clock rate}) \;/\; (\text{Instruction count} \times \text{CPI} \times 10^6)\\ &= \text{Clock rate} \;/\; (\text{CPI} \times 10^6)\end{aligned}$$

- Major problem with MIPS rating: As shown above the MIPS rating does not account for the count of instructions executed (I).
  - A higher MIPS rating in many cases may not mean higher performance or better execution time. i.e. due to compiler design variations.
- In addition the MIPS rating:
  - Does not account for the instruction set architecture (ISA) used.
    - Thus it cannot be used to compare computers/CPUs with different instruction sets.
  - Easy to abuse: Program used to get the MIPS rating is often omitted.
    - Often the Peak MIPS rating is provided for a given CPU which is obtained using a program comprised entirely of instructions with the lowest CPI for the given CPU design which does not represent real programs.

$$T = I \times CPI \times C$$

# Computer Performance Measures : MIPS (Million Instructions Per Second) Rating

- Under what conditions can the MIPS rating be used to compare performance of different CPUs?

- The MIPS rating is only valid to compare the performance of different CPUs provided that the following conditions are satisfied:

  1 The same program is used

  (actually this applies to all performance metrics)

  2 The same ISA is used

  3 The same compiler is used

  ⇒ (Thus the resulting programs used to run on the CPUs and obtain the MIPS rating are identical at the machine code level including the same instruction count)

  (binary)

# Compiler Variations, MIPS & Performance: An Example

- For a machine (CPU) with instruction classes:

| Instruction class | CPI |
|---|---|
| A | 1 |
| B | 2 |
| C | 3 |

- For a given high-level language program, two compilers produced the following executed instruction counts:

| | Instruction counts (in millions) for each instruction class | | |
|---|---|---|---|
| Code from: | A | B | C |
| Compiler 1 | 5 | 1 | 1 |
| Compiler 2 | 10 | 1 | 1 |

- The machine is assumed to run at a clock rate of 100 MHz.

# Compiler Variations, MIPS & Performance: An Example (Continued)

MIPS = Clock rate / (CPI x $10^6$) = 100 MHz / (CPI x $10^6$)

CPI = CPU execution cycles / Instructions count

$$CPU\ clock\ cycles = \sum_{i=1}^{n} \left( CPI_i \times C_i \right)$$

CPU time = Instruction count x CPI / Clock rate

- For compiler 1:
  - $CPI_1$ = (5 x 1 + 1 x 2 + 1 x 3) / (5 + 1 + 1) = 10 / 7 = 1.428
  - MIPS $Rating_1$ = 100 / (1.428 x $10^6$) = 70.0 MIPS
  - CPU $time_1$ = ((5 + 1 + 1) x $10^6$ x 1.428) / (100 x $10^6$) = 0.10 seconds
- For compiler 2:
  - $CPI_2$ = (10 x 1 + 1 x 2 + 1 x 3) / (10 + 1 + 1) = 15 / 12 = 1.25
  - MIPS $Rating_2$ = 100 / (1.25 x $10^6$) = 80.0 MIPS
  - CPU $time_2$ = ((10 + 1 + 1) x $10^6$ x 1.25) / (100 x $10^6$) = 0.15 seconds

MIPS rating indicates that compiler 2 is better while in reality the code produced by compiler 1 is faster

# Computer Performance Measures : MFLOPS (Million FLOating-Point Operations Per Second)

- A floating-point operation is an addition, subtraction, multiplication, or division operation applied to numbers represented by a single or a double precision floating-point representation.
- MFLOPS, for a specific program running on a specific computer, is a measure of millions of floating point-operation (megaflops) per second:

MFLOPS = Number of floating-point operations / (Execution time x $10^6$ )

- MFLOPS rating is a better comparison measure between different machines (applies even if ISAs are different) than the MIPS rating.
  - Applicable even if ISAs are different
- Program-dependent: Different programs have different percentages of floating-point operations present. i.e compilers have no floating- point operations and yield a MFLOPS rating of zero.
- Dependent on the type of floating-point operations present in the program.
  - Peak MFLOPS rating for a CPU: Obtained using a program comprised entirely of the simplest floating point instructions (with the lowest CPI) for the given CPU design which does not represent real floating point programs.

# Quantitative Principles of Computer Design

- Amdahl's Law:

  The performance gain from improving some portion of a computer is calculated by:

i.e using some enhancement

$$\text{Speedup} = \frac{\text{Performance for entire task using the enhancement}}{\text{Performance for the entire task without using the enhancement}}$$

$$\text{or Speedup} = \frac{\text{Execution time without the enhancement}}{\text{Execution time for entire task using the enhancement}}$$

Here: Task = Program

Recall: Performance = 1 /Execution Time

# Performance Enhancement Calculations: Amdahl's Law

- Amdahl's Law:

$$\text{Speedup(E)} = \frac{\text{Execution Time without enhancement}}{\text{Execution Time with enhancement}}$$

- Suppose a program contains s portion of code which requires sequential execution and and f portion of code can run in parallel fashion.
- We can write as s=1-f
- Suppose T be the execution time of program using single processor
- So, T= T(1-f)+ Tf
- If n processor will be used,
- So, T= T(1-f)+ Tf / N

$$\text{Speedup} = \frac{T(1-f) + Tf}{T(1-f) + \frac{Tf}{N}} = \frac{1}{(1-f) + \frac{f}{N}}$$

# Pictorial Depiction of Amdahl's Law

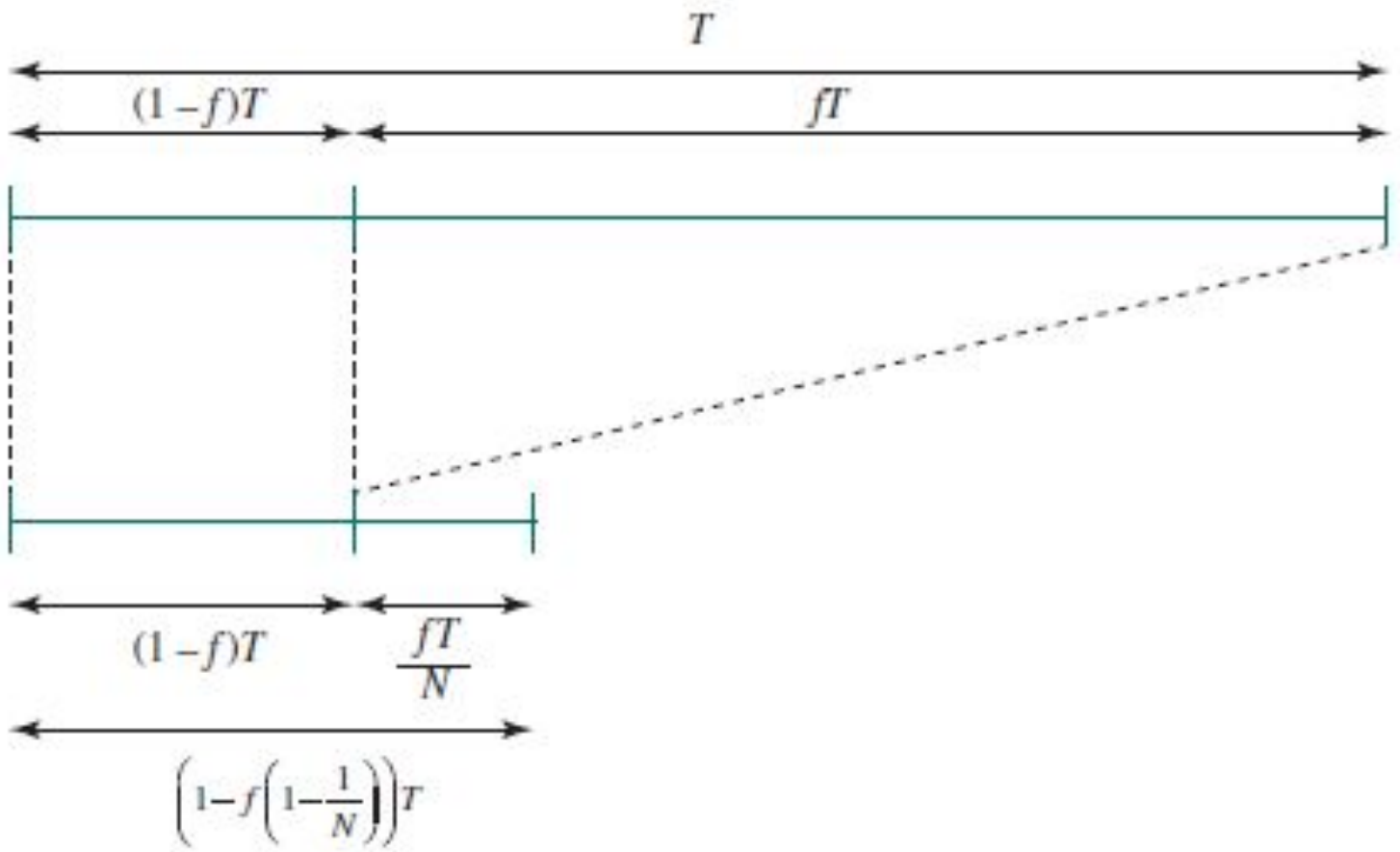
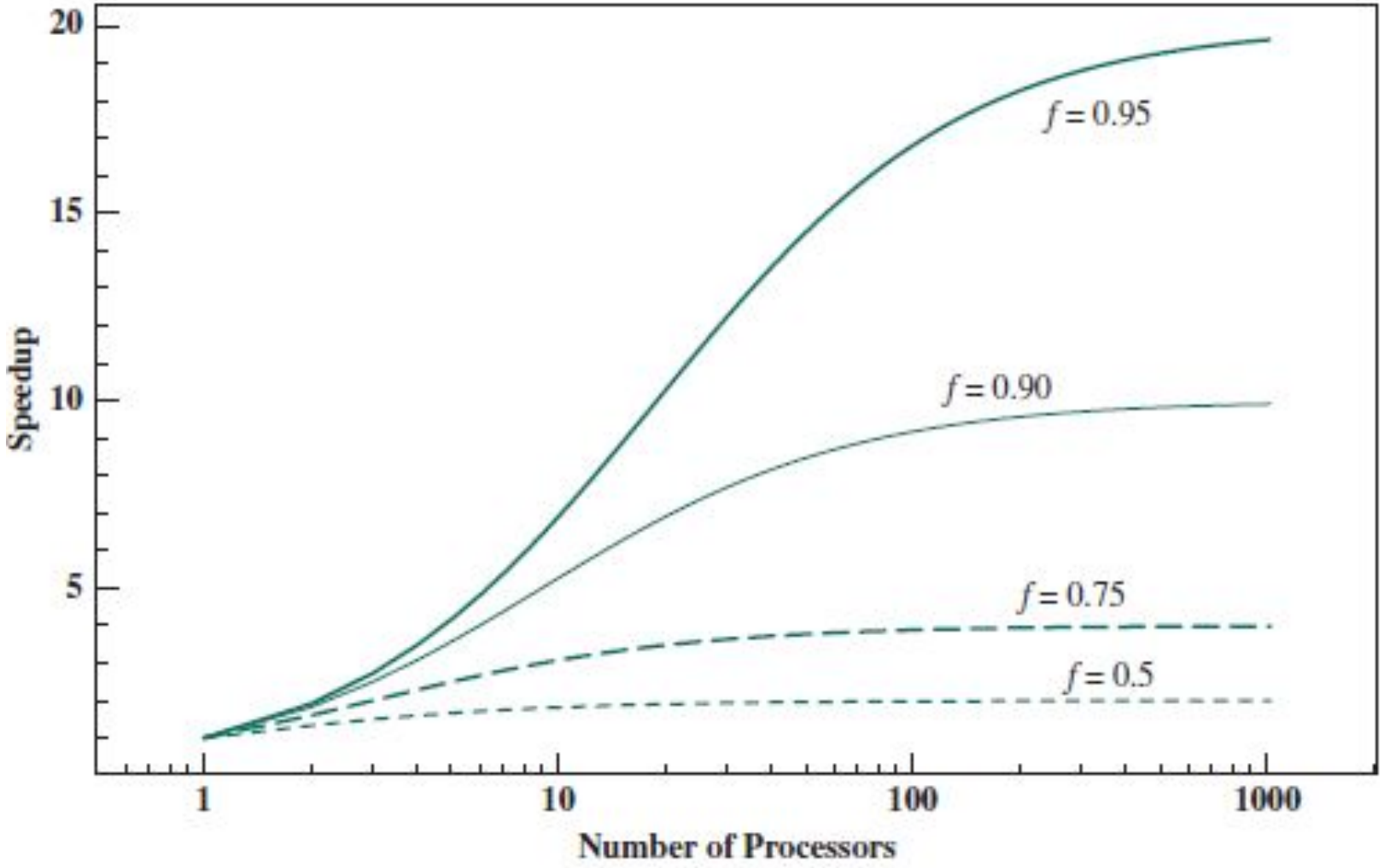



Illustration of Amdahl's Law [5]




Amdahl's Law for Multiprocessors [5]

# *Example*

- Suppose that a task makes extensive use of floating-point operations, with 40% of the time consumed by floating-point operations. With a new hardware design, the floating-point module is speed up by a factor of *4. Find overall Speedup?*

- *Solution: Floating point operation= 0.4*
- *Non floating point operation = 1-0.4=0.6*
- *Speedup = 1/ (0.6+0.4/4) =1.42*

# Performance Enhancement Example

- For the RISC machine with the following instruction mix given earlier:

| OP | Freq | Cycles | CPI(i) | % Time |
|---|---|---|---|---|
| ALU | 50% | 1 | .5 | 23% |
| Load | 20% | 5 | 1.0 | 45% |
| Store | 10% | 3 | .3 | 14% |
| Branch | 20% | 2 | .4 | 18% |

$$\frac{CPI_i \times F_i}{CPI}$$

CPI = 2.2

- If a CPU design enhancement improves the CPI of load instructions from 5 to 2, what is the resulting performance improvement from this enhancement:

Fraction enhanced = F = 45% or .45

Unaffected fraction = 1- F = 100% - 45% = 55% or .55

Factor of enhancement = S = 5/2 = 2.5

Using Amdahl's Law:

$$\text{Speedup(E)} = \frac{1}{(1 - F) + F/S} = \frac{1}{.55 + .45/2.5} = 1.37$$

# An Alternative Solution Using CPU Equation

| Op | Freq | Cycles | CPI(i) | % Time |
|---|---|---|---|---|
| ALU | 50% | 1 | .5 | 23% |
| Load | 20% | 5 | 1.0 | 45% |
| Store | 10% | 3 | .3 | 14% |
| Branch | 20% | 2 | .4 | 18% |

CPI = 2.2

- If a CPU design enhancement improves the CPI of load instructions from 5 to 2, what is the resulting performance improvement from this enhancement:

Old CPI = 2.2

New CPI of load is now 2 instead of 5

New CPI = .5 x 1 + .2 x 2 + .1 x 3 + .2 x 2 = 1.6

$$\text{Speedup(E)} = \frac{\text{Original Execution Time}}{\text{New Execution Time}} = \frac{\text{Instruction count} \times \text{old CPI} \times \text{clock cycle}}{\text{Instruction count} \times \text{new CPI} \times \text{clock cycle}}$$

$$= \frac{\text{old CPI}}{\text{new CPI}} = \frac{2.2}{1.6} = 1.37$$

Which is the same speedup obtained from Amdahl's Law in the first solution.

$$T = I \times CPI \times C$$

# Performance Enhancement Example

- A program runs in 100 seconds on a machine with multiply operations responsible for 80 seconds of this time. By how much must the speed of multiplication be improved to make the program four times faster?

$$\text{Speedup(E)} = \frac{1}{(1 - F) + F/S} = 4 = \frac{1}{(1 - .8) + .8/S} = \frac{1}{.2 + .8/s}$$

Solving for S gives S= 16

Hence multiplication should be 16 times faster to get an overall speedup of 4.



Machine = CPU

# Performance Enhancement Example

- For the previous example with a program running in 100 seconds on a machine with multiply operations responsible for 80 seconds of this time. By how much must the speed of multiplication be improved to make the program five times faster?

Speedup= 1 / (0.2 + (0.8/K))

5=1 / (0.2 + (0.8/K))

K+4 =K

4=0

No amount of multiplication speed improvement can achieve this.

# Extending Amdahl's Law To Multiple Enhancements

n enhancements each affecting a different portion of execution time

- Suppose that enhancement $E_i$ accelerates a fraction $F_i$ of the original execution time by a factor $S_i$ and the remainder of the time is unaffected then:

i = 1, 2, …. n

$$Speedup = \frac{\text{Original Execution Time}}{\left((1 - \sum_i F_i) + \sum_i \frac{F_i}{S_i}\right) X \text{Original Execution Time}}$$

Unaffected fraction

$$Speedup = \frac{1}{\left((1 - \sum_i F_i) + \sum_i \frac{F_i}{S_i}\right)}$$

What if the fractions given are after the enhancements were applied?
How would you solve the problem? (i.e find expression for speedup)

Note: All fractions $F_i$ refer to original execution time before the enhancements are applied.

# Amdahl's Law With Multiple Enhancements: Example

- Three CPU performance enhancements are proposed with the following speedups and percentage of the code execution time affected:

  $\text{Speedup}_1 = S_1 = 10$ $\quad$ $\text{Percentage}_1 = F_1 = 20\%$
  $\text{Speedup}_2 = S_2 = 15$ $\quad$ $\text{Percentage}_1 = F_2 = 15\%$
  $\text{Speedup}_3 = S_3 = 30$ $\quad$ $\text{Percentage}_1 = F_3 = 10\%$

- While all three enhancements are in place in the new design, each enhancement affects a different portion of the code and only one enhancement can be used at a time.
- What is the resulting overall speedup?

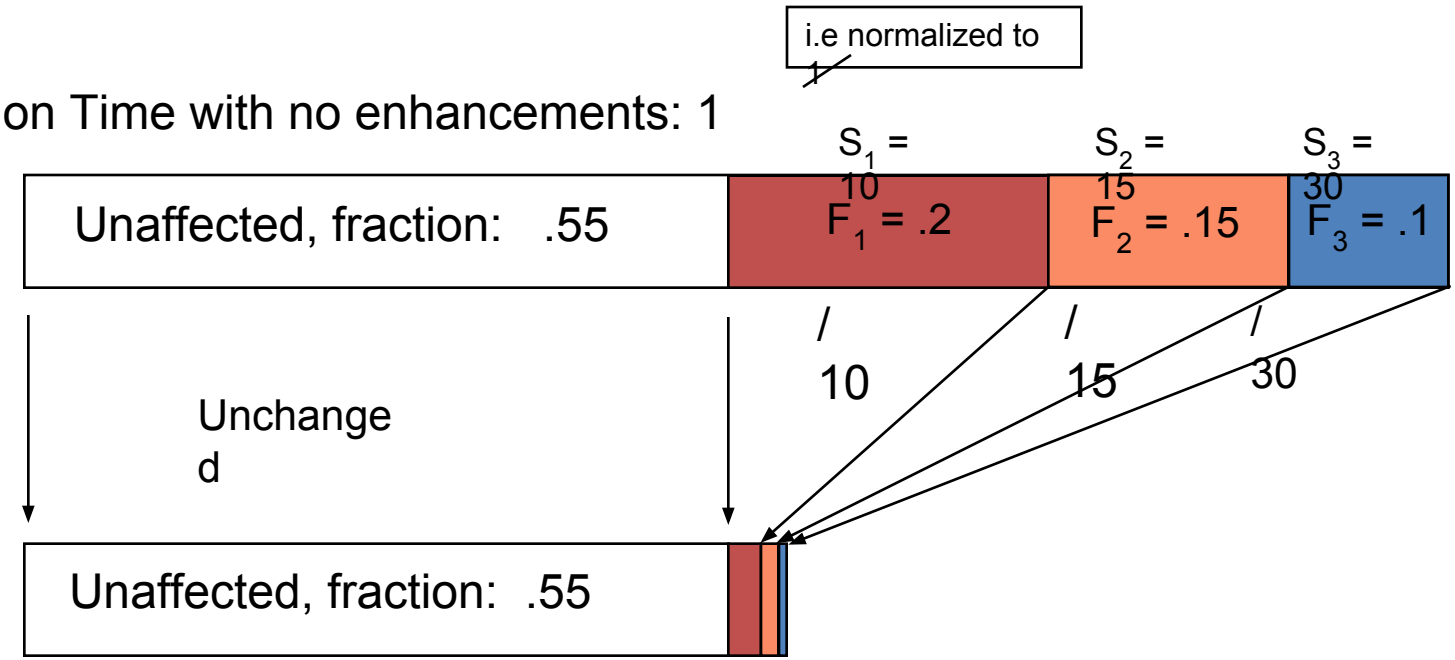
$$\textbf{\textit{Speedup}} = \frac{1}{\left((1 - \sum_i \boldsymbol{F}_i) + \sum_i \frac{\boldsymbol{F}_i}{\boldsymbol{S}_i}\right)}$$

- Speedup = 1 / [(1 - .2 - .15 - .1) + .2/10 + .15/15 + .1/30)]
  = 1 / [ .55 + .0333 ]
  = 1 / .5833 = 1.71

# Pictorial Depiction of Example

Before:

Execution Time with no enhancements: 1




After:

Execution Time with enhancements: .55 + .02 + .01 + .00333 = .5833

Speedup = 1 / .5833 = 1.71

What if the fractions given are after the enhancements were applied?
How would you solve the problem?

Note: All fractions $F_i$ refer to original execution time.

# That's all for now…

# References


- [1]. Cramming more components onto integrated circuits, Reprinted from Electronics, volume 38, number 8, April 19, 1965, pp.114 ff.
- [2]. https://www.top500.org/lists/top500/2020/06/highs
- [3]. M Morris R. Mano, “Computer System Architecture”, 3rd edition, Pearson Education
- [4] Computer Organization and Design THE HARDWARE / SOFTWAREINTERFACE David A. Patterson University of California, Berkeley John L. Hennessy Stanford University
- [5] Computer Organization and Architecture *Designing for Performance* Tenth Edition William Stallings