# CSE 360-Computer Architecture

## Lecture-2

**Dr. Shamim Akhter**

Associate Professor

Department of Computer Science and Engineering

email: shamimakhter@ewubd.edu

When would you want to compare performance between different computers?

# Performance Metrics

- Purchasing perspective
  - given a collection of machines, which has the
    - best performance ?
    - least cost ?
    - best cost/performance?
- Design perspective
  - faced with design options, which has the
    - best performance improvement ?
    - least cost ?
    - best cost/performance?
- Both require
  - basis for comparison
  - metric for evaluation

- Our goal is to understand what factors in the architecture contribute
  - to overall system performance
  - relative importance (and cost) of these factors.

What ways can be used to determine perform on a desktop PC?

What ways can be used to determine perform on a server?

# Computer Performance: TIME, TIME, TIME!!!

- *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 *I* wait for the database query?

Individual user concerns…

- *Throughput*:
  - how *many* jobs can the machine run at once?
  - what is the *average* execution rate?
  - how *much* work is getting done?

Systems manager concerns…

- *If we upgrade a machine with a new processor what do we increase?*
- *If we add a new machine to the lab what do we increase?*

- What ways can be used to determine perform on a desktop PC?
- What ways can be used to determine perform on a server?

# Throughput and Response Time

- Q1. If we upgrade a machine with a (faster) processor, what do we increase?
- Q2. Adding additional processors to a system that uses multiple processors for separate tasks-searching web

- Q1 Answer: Response Time Decrease->Improve throughput

- Q2 Answer: Only throughput increase.

  process stores @ Job Queue then improve

  response time as reducing wait time

# Execution Time

- **Elapsed Time**
  - counts everything (*disk and memory accesses, waiting for I/O, running other programs, etc.*) from start to finish
  - a useful number, but often not good for comparison purposes

  elapsed time = CPU time + wait time (I/O, other programs, etc.)

Time spent in user space

Time spent in kernel space

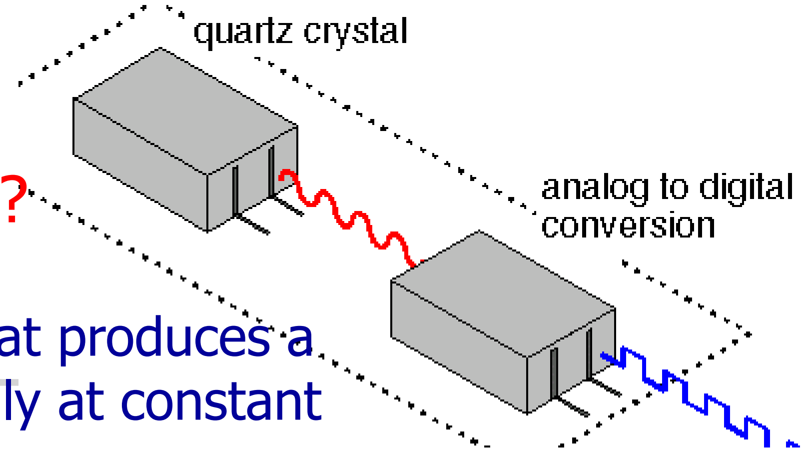
- **CPU time**
  - doesn't count waiting for I/O or time spent running other programs
  - can be divided into *user CPU time* and *system CPU time* (OS calls)

  CPU time = user CPU time + system CPU time

$\Rightarrow$ elapsed time = user CPU time + system CPU time + wait time

- Our focus: *user CPU time* (*CPU execution time* or, simply, *execution time*)
  - time spent executing the lines of code that are *in our program*

# Clock Cycles




How do you measure execution time?

X seconds

But cycles.

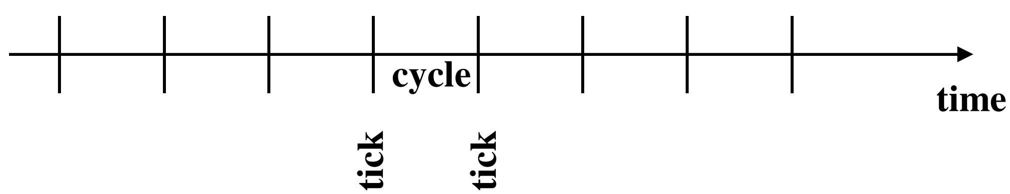
A clock is a circuit that produces a periodic signal, usually at constant frequency or rate.

Hardware events progress cycle by cycle:

- in other words, each event, e.g., multiplication, addition, etc., is a sequence of cycles

$$\frac{\text{seconds}}{\text{program}} = \frac{\text{cycles}}{\text{program}} \times \frac{\text{seconds}}{\text{cycle}}$$

- *Clock ticks* indicate start and end of cycles:




- *cycle time* = time between ticks = seconds per cycle

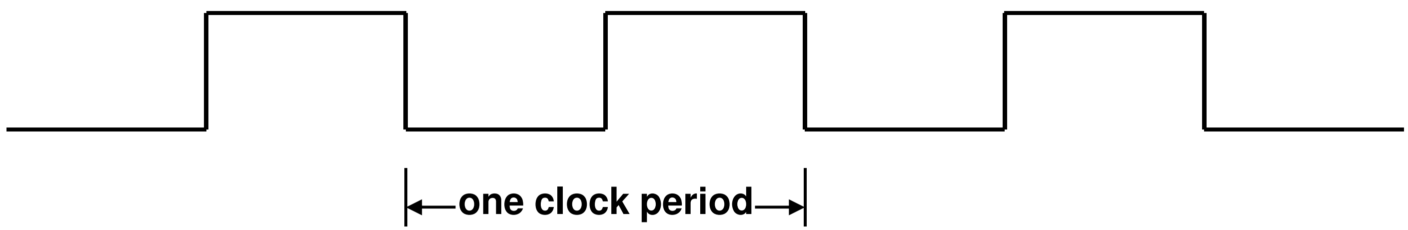
NB: Clock ticks =smallest unit of time recognized by a device.

Ex. 66MHz = 66 million clock ticks per second.

# Clock Rate

- Clock rate (MHz, GHz) is inverse of clock cycle time (clock period)

$$CC = 1 / CR$$




clock rate (frequency) = cycles per second (1 Hz. = 1 cycle/sec,
1 MHz. = $10^6$ cycles/sec)

**Example:** A 200 Mhz. clock has a cycle time

$$\frac{1}{200\times10^{6}}\times10^{9} = 5 \text{ nanosecor}$$

# Definition of Performance

**Improves by reducing execution time**

For some program running on machine X:

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

*X is n times faster than Y* means:

$$\text{Performance}_X \; / \; \text{Performance}_Y = n$$

# Performance Equation I

$$\frac{\text{seconds}}{\text{program}} = \frac{\text{cycles}}{\text{program}} \times \frac{\text{seconds}}{\text{cycle}}$$

equivalently

$$\begin{array}{l}\text{CPU execution time}\\ \text{for a program}\end{array} = \begin{array}{l}\text{CPU clock cycles}\\ \text{for a program}\end{array} \times \text{Clock cycle time}$$

- So, to improve performance one can either:
  - reduce execution time
  - reduce program cycles, or
  - reduce the clock cycle time, or, equivalently, increase the clock rate

# Example

- Our favorite program runs in 10 seconds on computer A, which has a 400Mhz. clock.
- We are trying to help a computer designer to build a new machine B, that will run this program in 6 seconds. The designer can use new (or perhaps more expensive) technology to substantially increase the clock rate, but has informed us that this increase will affect the rest of the CPU design, causing machine B to require 1.2 times as many clock cycles as machine A for the same program.

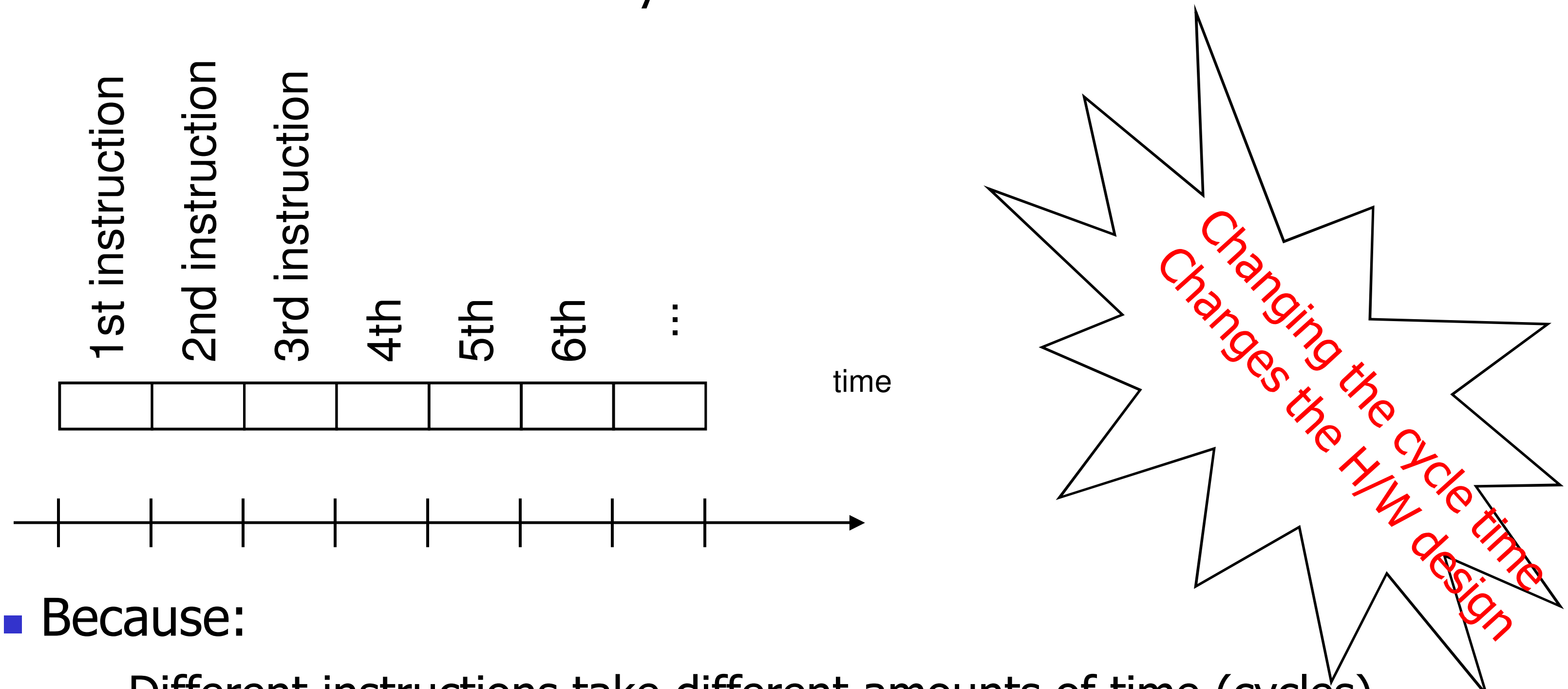
- *What clock rate should we tell the designer to target?*

- Can we predict the execution time, without running in CPU?

- Assume: # of instruction = # of cycles

# Assumption is incorrect!

- Could assume that # of cycles = # of instructions




- Because:
  - Different instructions take different amounts of time (cycles)
  - Multiplication takes more time than addition
  - Floating point operations take longer than integer ones
  - Accessing memory takes more time than accessing registers

They have different execution steps [Datapath]

# Solution

- Execution time is equals to
- # of instructions executed X the average time per instruction

# Performance Equation II

$$\text{CPU execution time for a program} = \text{CPU clock cycles for a program} \times \text{Clock cycle time}$$

$$\#\text{ CPU clock cycles for a program} = \#\text{ Instructions for a program} \times \text{Average clock cycles per instruction}$$

$$\text{CPU execution time for a program} = \text{Instruction count for a program} \times \text{average CPI} \times \text{Clock cycle time}$$

- **Clock cycles per instruction** (CPI) – the average number of clock cycles each instruction takes to execute
  - A way to compare two different implementations of the same ISA

| | CPI for this instruction class | | |
|---|---|---|---|
| | A | B | C |
| CPI | 1 | 2 | 3 |

# CPI Example I

- Suppose we have two implementations of the same instruction set architecture (ISA). For some program:
  - machine A has a clock cycle time of 10 ns. and a CPI of 2.0
  - machine B has a clock cycle time of 20 ns. and a CPI of 1.2

- Which machine is faster for this program, and by how much?

# Effective CPI

- Computing the overall effective CPI is done by looking at the different types of instructions and their individual cycle counts and averaging

$$\text{Overall effective CPI} = \sum_{i=1}^{n} (CPI_i \times C_i)$$

- Where $C_i$ is the count **(percentage)** of the number of instructions of class i executed
  - $C_i$ not in %, the above equation will be divided by total Ic
- $CPI_i$ is the (average) number of clock cycles per instruction for that instruction class
- n is the number of instruction classes

$$CPI_{effective} = (C1 \times CPI1 + C2 \times CPI2 + \ldots\ Cn \times CPIn) / (c1+c2+\ldots+cn)$$

$$= C1 \times CPI1/T + C2 \times CPI2/T \ldots\ldots\ldots\ldots CnxCPIn/T$$

Clock Cycle Time $= I \times CPI_{effective}$ from performance eqn II

# A Simple Example – calculate average CPI

| Op | Freq | $CPI_i$ | Freq x $CPI_i$ |
|---|---|---|---|
| ALU | 50% | 1 | |
| Load | 20% | 5 | |
| Store | 10% | 3 | |
| Branch | 20% | 2 | |
| Overall effective CPI | | | $\Sigma =$ |

# A Simple Example

| Op | Freq | $CPI_i$ | Freq x $CPI_i$ |
|---|---|---|---|
| ALU | 50% | 1 | .5 |
| Load | 20% | 5 | 1.0 |
| Store | 10% | 3 | .3 |
| Branch | 20% | 2 | .4 |
| | | Overall effective CPI | $\Sigma =$ 2.2 |

# A Simple Example

| Op | Freq | $CPI_i$ | Freq x $CPI_i$ |
|---|---|---|---|
| ALU | 50% | 1 | |
| Load | 20% | 5 | |
| Store | 10% | 3 | |
| Branch | 20% | 2 | |
| | | Overall effective CPI | $\Sigma =$ |

- How much faster would the machine be if a better data cache reduced the average load time to 2 cycles?
- How does this compare with using branch prediction to save a cycle off the branch time?
- What if two ALU instructions could be executed at once?

# A Simple Example

| Op | Freq | $CPI_i$ | Freq x $CPI_i$ | Q1 |
|---|---|---|---|---|
| ALU | 50% | 1 | .5 | .5 |
| Load | 20% | 5 | 1.0 | .4 |
| Store | 10% | 3 | .3 | .3 |
| Branch | 20% | 2 | .4 | .4 |
| | | Overall effective CPI | $\Sigma =$ 2.2 | 1.6 |

- How much faster would the machine be if a better data cache reduced and replaced the average load time to 2 cycles?

# A Simple Example

| Op | Freq | $CPI_i$ | Freq x $CPI_i$ | Q1 |
|---|---|---|---|---|
| ALU | 50% | 1 | .5 | .5 |
| Load | 20% | 5 | 1.0 | .4 |
| Store | 10% | 3 | .3 | .3 |
| Branch | 20% | 2 | .4 | .4 |
| Overall effective CPI | | | $\Sigma =$ 2.2 | 1.6 |

- How much faster would the machine be if a better data cache reduced and replaced the average load time to 2 cycles?

CPU time new = 1.6 x IC x CCT so 2.2/1.6 means (1.375 times) 37.5% faster

(Notice that 2.2 x IC x CCT / 1.6 x IC x CCT = 2.2/1.6, since the IC and CCT

remain unchanged by adding a bigger cache)

# A Simple Example

| Op | Freq | $CPI_i$ | Freq x $CPI_i$ | Q2 |
|---|---|---|---|---|
| ALU | 50% | 1 | .5 | .5 |
| Load | 20% | 5 | 1.0 | 1.0 |
| Store | 10% | 3 | .3 | .3 |
| Branch | 20% | 2 | .4 | .2 |
| Overall effective CPI | | | $\Sigma =$ 2.2 | 2.0 |

- How does this compare with using branch prediction to save a cycle off the branch time?

# A Simple Example

| Op | Freq | $CPI_i$ | Freq x $CPI_i$ | Q2 |
|---|---|---|---|---|
| ALU | 50% | 1 | .5 | .5 |
| Load | 20% | 5 | 1.0 | 1.0 |
| Store | 10% | 3 | .3 | .3 |
| Branch | 20% | 2 | .4 | .2 |
| Overall effective CPI | | | $\Sigma =$ 2.2 | 2.0 |

- How does this compare with using branch prediction to save a cycle off the branch time?

CPU time new = 2.0 x IC x CCT so 2.2/2.0 means (1.1 times)10% faster

# A Simple Example

| Op | Freq | $CPI_i$ | Freq x $CPI_i$ | Q3 |
|---|---|---|---|---|
| ALU | 50% | 1 | .5 | .25 |
| Load | 20% | 5 | 1.0 | 1.0 |
| Store | 10% | 3 | .3 | .3 |
| Branch | 20% | 2 | .4 | .4 |
| Overall effective CPI | | | $\Sigma =$ 2.2 | 1.95 |

- What if two ALU instructions could be executed at once?

# A Simple Example

| Op | Freq | $CPI_i$ | Freq x $CPI_i$ | Q3 |
|---|---|---|---|---|
| ALU | 50% | 1 | .5 | .25 |
| Load | 20% | 5 | 1.0 | 1.0 |
| Store | 10% | 3 | .3 | .3 |
| Branch | 20% | 2 | .4 | .4 |
| Overall effective CPI | | | $\Sigma =$ 2.2 | 1.95 |

- What if two ALU instructions could be executed at once?

CPU time new = 1.95 x IC x CCT so 2.2/1.95 means (1.128 times) 12.8% faster

# A Simple Example

| Op | Freq | $CPI_i$ | Freq x $CPI_i$ | Q1 | Q2 | Q3 |
|---|---|---|---|---|---|---|
| ALU | 50% | 1 | .5 | .5 | .5 | .25 |
| Load | 20% | 5 | 1.0 | .4 | 1.0 | 1.0 |
| Store | 10% | 3 | .3 | .3 | .3 | .3 |
| Branch | 20% | 2 | .4 | .4 | .2 | .4 |
| Overall effective CPI | | | $\Sigma =$ 2.2 | 1.6 | 2.0 | 1.95 |

- How much faster would the machine be if a better data cache reduced and replaced the average load time to 2 cycles?

  CPU time new = 1.6 x IC x CCT so 2.2/1.6 means 37.5% faster

- How does this compare with using branch prediction to save a cycle off the branch time?

  CPU time new = 2.0 x IC x CCT so 2.2/2.0 means 10% faster

- What if two ALU instructions could be executed at once?

  CPU time new = 1.95 x IC x CCT so 2.2/1.95 means 12.8% faster

# CPI Example II

- A compiler designer is trying to decide between two code sequences for a particular machine. The hardware designers have supplied the following facts:

| | CPI for this instruction class | | |
|---|---|---|---|
| | A | B | C |
| CPI | 1 | 2 | 3 |

- For a particular high-level-language statement, the compiler writer is considering two code sequences that requires the following instruction counts:

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

- Which sequence will be faster? How much? What is the CPI for each sequence?

# MIPS Rate
# (Millions of Instructions per Second )

$$\text{MIPS Rate} = \frac{Ic}{T \times 10^6} = \frac{f}{CPI \times 10^6}$$

$$\text{MFLOPS Rate} = \frac{\text{Number of executed floating - point operations in a program}}{\text{Execution time} \times 10^6}$$

# Example III

Consider the execution of a program that results in the execution of 2 million instructions on a 400-MHz processor. The program consists of four major types of instructions. The instruction mix and the CPI for each instruction type are given below based on the result of a program trace experiment:

| **Instruction Type** | **CPI** | **Instruction Mix (%)** |
|---|---|---|
| Arithmetic and logic | 1 | 60 |
| Load/store with cache hit | 2 | 18 |
| Branch | 4 | 12 |
| Memory ref | 8 | 10 |

Find the MIPS rate?

# Dependency

| | |
|---|---|
| **Performance** | **Depends on the algorithm, programming language, Compiler, ISA** |
| Execution Time | Depends on instruction counts, CPI, Clock Cycle time |
| Instruction Counts | Depends on ISA, Programmer effective coding, Compiler code optimization |
| CPI | ISA, H/W Organization |
| Clock Cycle Time | H/W Organization, Implementation strategies |

# Amdahl's Law

## Potential speedup using multiple processors environment

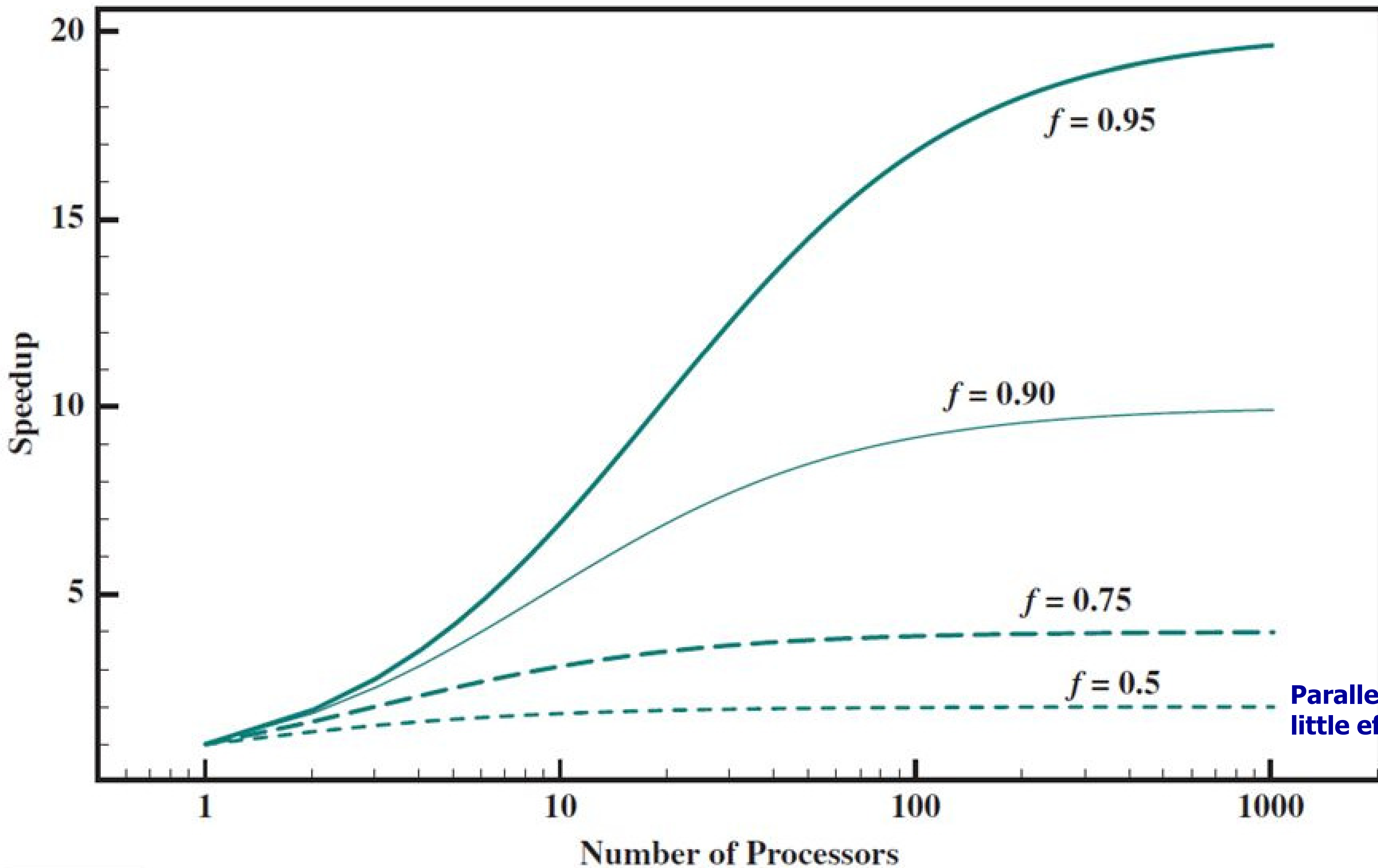
- Consider a program running on a single processor such that a fraction *(1-f)* of the execution time involves code that is inherently serial and a fraction *f* that involves code that is infinitely parallelizable with no scheduling overhead. T is the total execution time of the serial program. Then Speedup using N processors is as follows:

$$\text{Speedup} = \frac{\text{Time to execute program on a single processor}}{\text{Time to execute program on N parallel processors}}$$

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

N -> Inf
Height possible speed up
bound to (1-f)

# Amdahl’s Law for Multiprocessors




**Parallelism has little effect**