



CSCI-GA.3033-012  
**Multicore Processors:**  
**Architecture & Programming**

## **Lecture 9: Performance Evaluation**

Mohamed Zahran (aka Z)  
mzahran@cs.nyu.edu  
<http://www.mzahran.com>



# What Are We trying to do?

- Measure, Report, and Summarize Performance
- Make intelligent choices

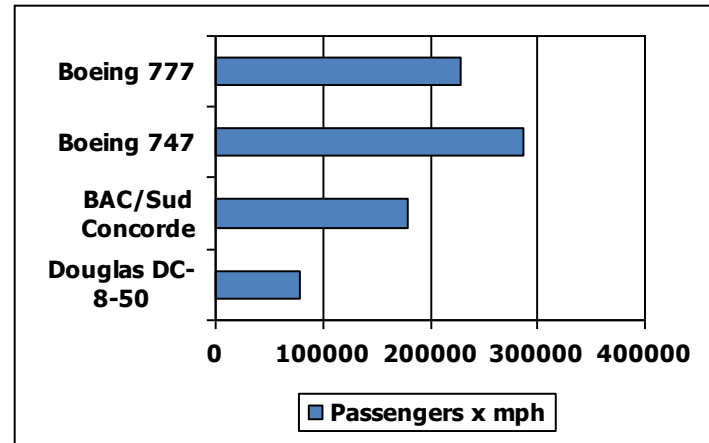
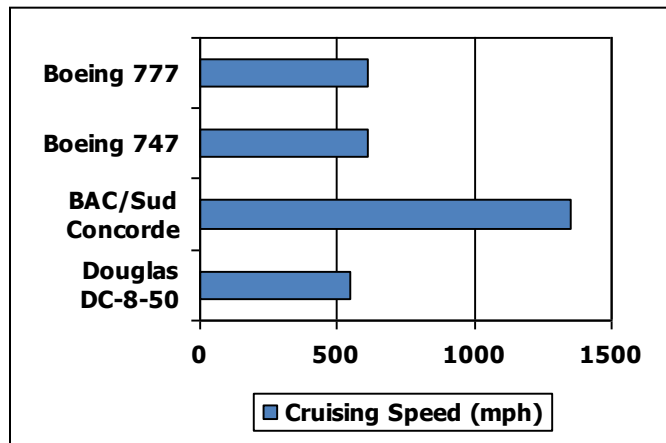
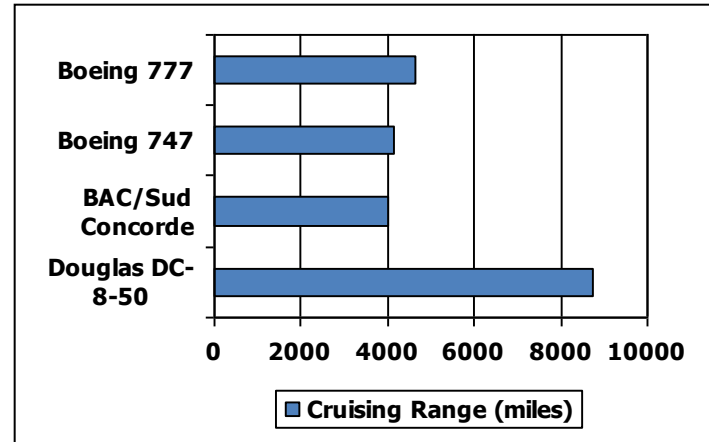
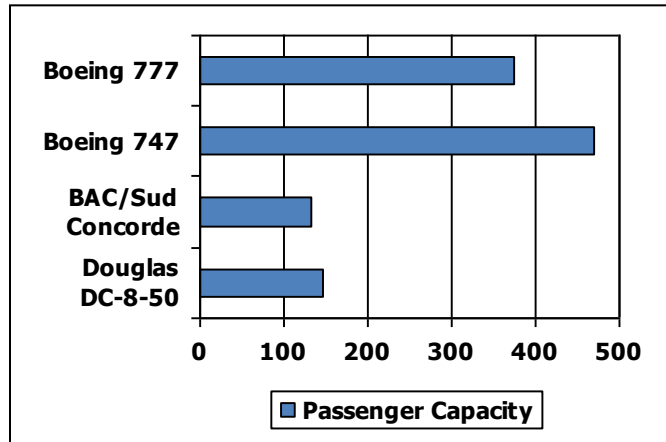
*Why is some hardware better than others for different programs?*

*What factors of system performance are hardware related?  
(e.g., Do we need a new machine, or a new operating system?)*

*Does performance measure depends on application type?*

# Defining Performance

- Which airplane has the best performance?



# Let's Start with Two Simple Metrics

- Response time (aka Execution Time)
  - The time between the start and completion of a task
- Throughput
  - Total amount of work done in a given time

What is the relationship between execution time and throughput?

# Computer Performance:

## TIME, TIME, TIME

- Response Time (latency)
  - How long does it take for my job to run?
  - How long does it take to execute a job?
  - How long must I wait for the database query?
- Throughput
  - How many jobs can the machine run at once?
  - What is the average execution rate?
  - How much work is getting done?

# Try to solve this...

- Do the following changes to the computer system increase throughput, decrease response time, or both?
  - Replacing the processor with a faster version
  - Adding additional processors to a system that uses multiple processors for separate tasks.

# Execution Time

- **Elapsed Time**

- counts everything (*disk and memory accesses, I/O, etc.*)
- a useful number, but often not good for comparison purposes

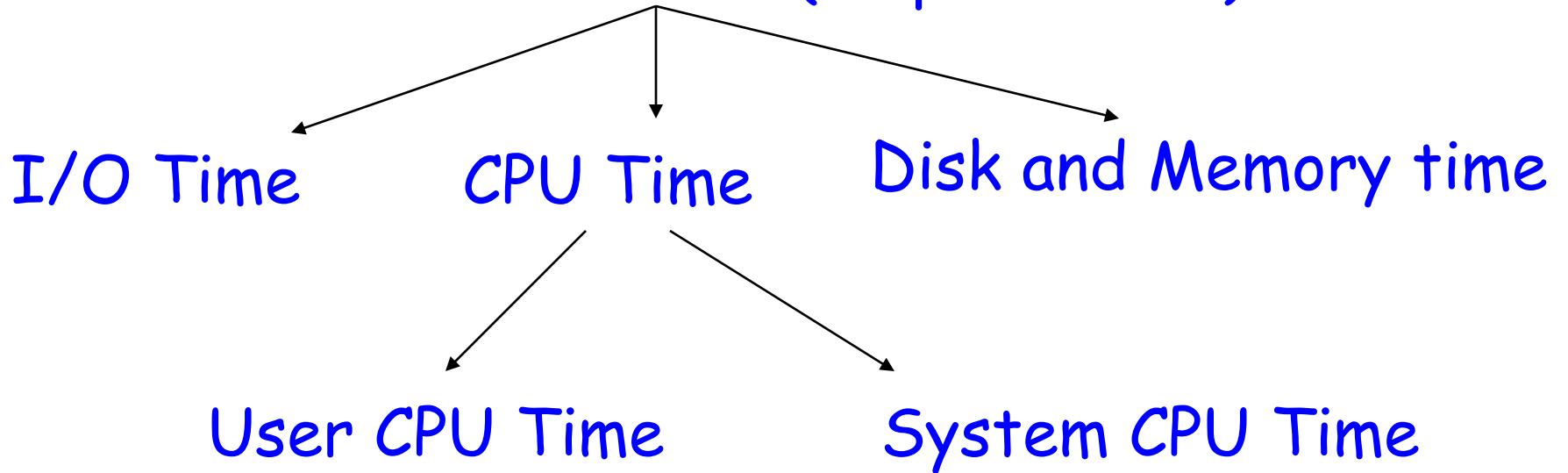
- **CPU time**

- doesn't count I/O or time spent running other programs
- can be broken up into system time, and user time

- **Our focus: user CPU time**

- time spent executing the lines of code that are "in" our program

# Execution Time (Elapsed Time)





# Book's Definition of Performance

- For some program running on machine X,

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

- "X is n times faster than Y"

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

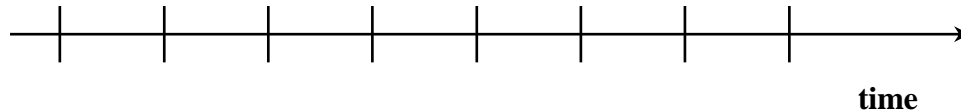
- Example: time taken to run a program
  - 10s on A, 15s on B
  - $\text{Execution Time}_B / \text{Execution Time}_A$   
 $= 15s / 10s = 1.5$
  - So A is 1.5 times faster than B

# Clock Cycles

- Instead of reporting execution time in seconds, we often use cycles

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

- Clock "ticks" indicate when to start activities (one abstraction):



- cycle time = time between ticks = seconds per cycle
- clock rate (frequency) = cycles per second (1 Hz. = 1 cycle/sec)
- A 4 Ghz. clock has a  $\frac{1}{4 \times 10^9} \times 10^{12} = 250$  picoseconds (ps) cycle time

# How to Improve Performance

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

So, to improve performance (everything else being equal) you can either (increase or decrease?)

- the # of required cycles for a program,
- the clock cycle time or, said another way, the clock rate.

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

ET

IC \* CPI

CT

$$ET = IC \times CPI \times CT$$

ET = Execution Time

CPI = Cycles Per Instruction

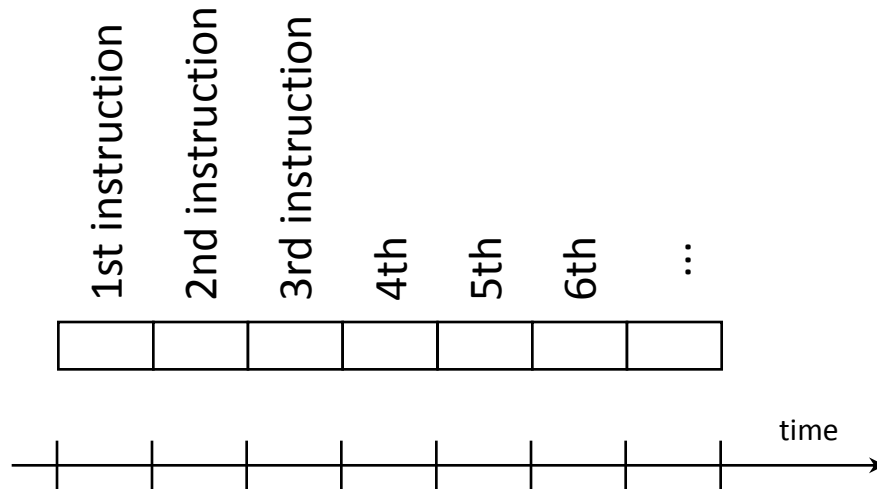
IC = Instruction Count

# An Interesting Question

- *If two machines have the same ISA which of our quantities (e.g., clock rate, CPI, execution time, # of instructions, MIPS) will always be identical?*

# How many cycles are required for a program?

- Could assume that number of cycles equals number of instructions

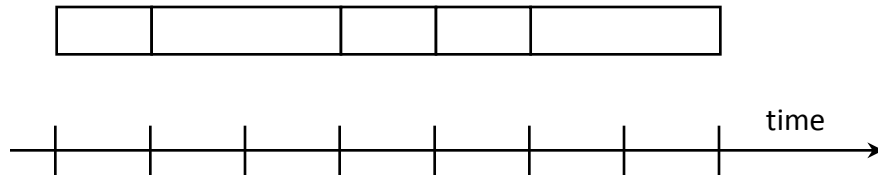


*This assumption is incorrect,*

*different instructions take different amounts of time on different machines.*

*Why? hint: remember that these are machine instructions, not lines of C code*

# Different numbers of cycles for different instructions



- Multiplication takes more time than addition
- Floating point operations take longer than integer ones
- Accessing memory takes more time than accessing registers
- *Important point: changing the cycle time often changes the number of cycles required for various instructions*

# Example

- Our favorite program runs in 10 seconds on computer A, which has a 4 GHz. clock. We are trying to help a computer designer 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?“

$$ET = IC * CPI * CT = \text{total\_cycles} * CT$$

$$A: 10 = \text{total\_cycles} * (1/4\text{GHz})$$

$$B: 6 = 1.2 * \text{total\_cycles} * (1/\text{clock\_rate})$$

$$10/6 = \text{clock\_rate}/(1.2 * 4\text{GHz})$$

$$\text{Clock\_rate} = 8\text{GHz}$$



# Now that we understand cycles

- A given program will require
  - some number of instructions (machine instructions)
  - some number of cycles
  - some number of seconds
- We have a vocabulary that relates these quantities:
  - cycle time (seconds per cycle)
  - clock rate (cycles per second)
  - CPI (cycles per instruction)
    - a floating point intensive application might have a higher CPI*
  - MIPS (millions of instructions per second)
    - this would be higher for a program using simple instructions*

# Performance

- Performance is determined by execution time
- Do any of the other variables equal performance?
  - # of cycles to execute program?
  - # of instructions in program?
  - # of cycles per second?
  - average # of cycles per instruction?
  - average # of instructions per second?

# CPI Example

- Suppose we have two implementations of the same instruction set architecture (ISA).

For some program,

Machine A has a clock cycle time of 250 ps and a CPI of 2.0

Machine B has a clock cycle time of 500 ps and a CPI of 1.2

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

[  $10^{-3}$  = milli,  $10^{-6}$  = micro,  $10^{-9}$  = nano,  $10^{-12}$  = pico,  $10^{-15}$  = femto ]

$$ET = IC * CPI * CT$$

$$ET_A = IC * 2 * 250 = 500IC$$

$$ET_B = IC * 1.2 * 500 = 600IC$$

# #Instructions Example

- A compiler designer is trying to decide between two code sequences for a particular machine. Based on the hardware implementation, there are three different classes of instructions: Class A, Class B, and Class C, and they require one, two, and three cycles (respectively).

The first code sequence has 5 instructions:  
2 of A, 1 of B, and 2 of C

The second sequence has 6 instructions:  
4 of A, 1 of B, and 1 of C.

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

# MIPS Example

- Two different compilers are being tested for a 4 GHz. machine with three different classes of instructions: Class A, Class B, and Class C, which require one, two, and three cycles (respectively). Both compilers are used to produce code for a large piece of software.

The first compiler's code uses 5 million Class A instructions, 1 million Class B instructions, and 1 million Class C instructions.

The second compiler's code uses 10 million Class A instructions, 1 million Class B instructions, and 1 million Class C instructions.

- Which sequence will be faster according to MIPS?
- Which sequence will be faster according to execution time?

# For Multithreaded Programs

- Shall we use execution time or throughput? or both?
- IPC is not accurate here
  - small timing variations may lead to different execution
  - Order at which threads enter critical section may vary
  - Different interrupt timing may lead to different scheduling decisions

The total number of instructions executed may be different across different runs!

# For Multithreaded Programs

The total number of instructions executed may be different across different runs!



**This effect increases with  
the number of cores**

System-level code account for a significant fraction of the total execution time

# Your Program Does Not Run in A Vacuum

- System software at least is there
- Multi-programming setting is very common in multicore settings
- Independent programs affect each other performance (why?)



# Some Metrics About Multiprogramming

Normalized progress of program  $i$   $\rightarrow$   $NP_i = \frac{T_i^{SP}}{T_i^{MP}}$

$T_i^{SP}$   $\leftarrow$  Time when running in isolation

$T_i^{MP}$   $\leftarrow$  Time when running with other programs



System throughput  $\rightarrow$

$$STP = \sum_{i=1}^n NP_i = \sum_{i=1}^n \frac{T_i^{SP}}{T_i^{MP}}$$

Higher-is-better metric

# Some Metrics About Multiprogramming

Normalized Turnaround time of program  $i$   $\longrightarrow$   $NTT_i = \frac{T_i^{MP}}{T_i^{SP}}$

Time when running with other programs  $\longleftarrow T_i^{MP}$

Time when running in isolation  $\longleftarrow T_i^{SP}$



Average normalized turnaround time  $\longrightarrow$

$$ANTT = \frac{1}{n} \sum_{i=1}^n NTT_i = \frac{1}{n} \sum_{i=1}^n \frac{T_i^{MP}}{T_i^{SP}},$$

Lower-is-better metric

# Other Metrics

~~$$IPC_{throughput} = \sum_{i=1}^n IPC_i$$~~


$$weighted\_speedup = \sum_{i=1}^n \frac{IPC_i^{MP}}{IPC_i^{SP}}$$

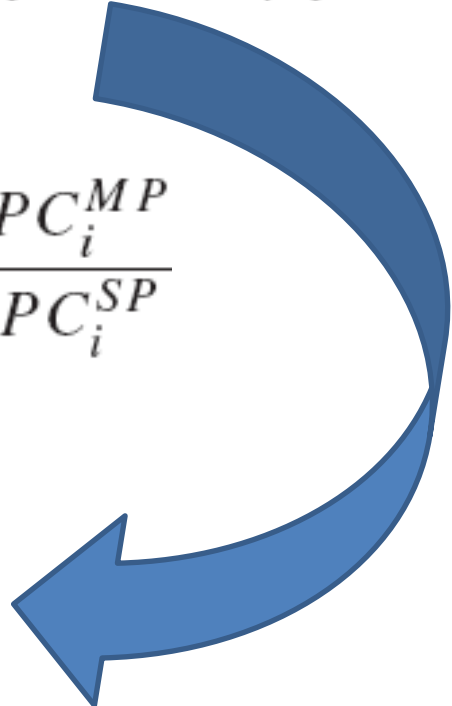
$$hmean = \frac{n}{\sum_{i=1}^n \frac{IPC_i^{SP}}{IPC_i^{MP}}}$$

# Other Metrics

$$STP = \sum_{i=1}^n NP_i = \sum_{i=1}^n \frac{T_i^{SP}}{T_i^{MP}}$$

$$ANTT = \frac{1}{n} \sum_{i=1}^n NTT_i = \frac{1}{n} \sum_{i=1}^n \frac{T_i^{MP}}{T_i^{SP}},$$


$$weighted\_speedup = \sum_{i=1}^n \frac{IPC_i^{MP}}{IPC_i^{SP}}$$

$$hmean = \frac{n}{\sum_{i=1}^n \frac{IPC_i^{SP}}{IPC_i^{MP}}}$$


# Important

What we saw for multiprogramming, can it be used in multithreading?

# Conclusions

- Performance evaluation is very important to assess programming quality as well as the underlying architecture and how they interact.
- IPC (or CPI) is not a good measure for multithreaded application