## Lecture 4:

# Performance (Redux)

**Introduction to Computer Architecture**
**UC Davis EEC 170, Fall 2019**

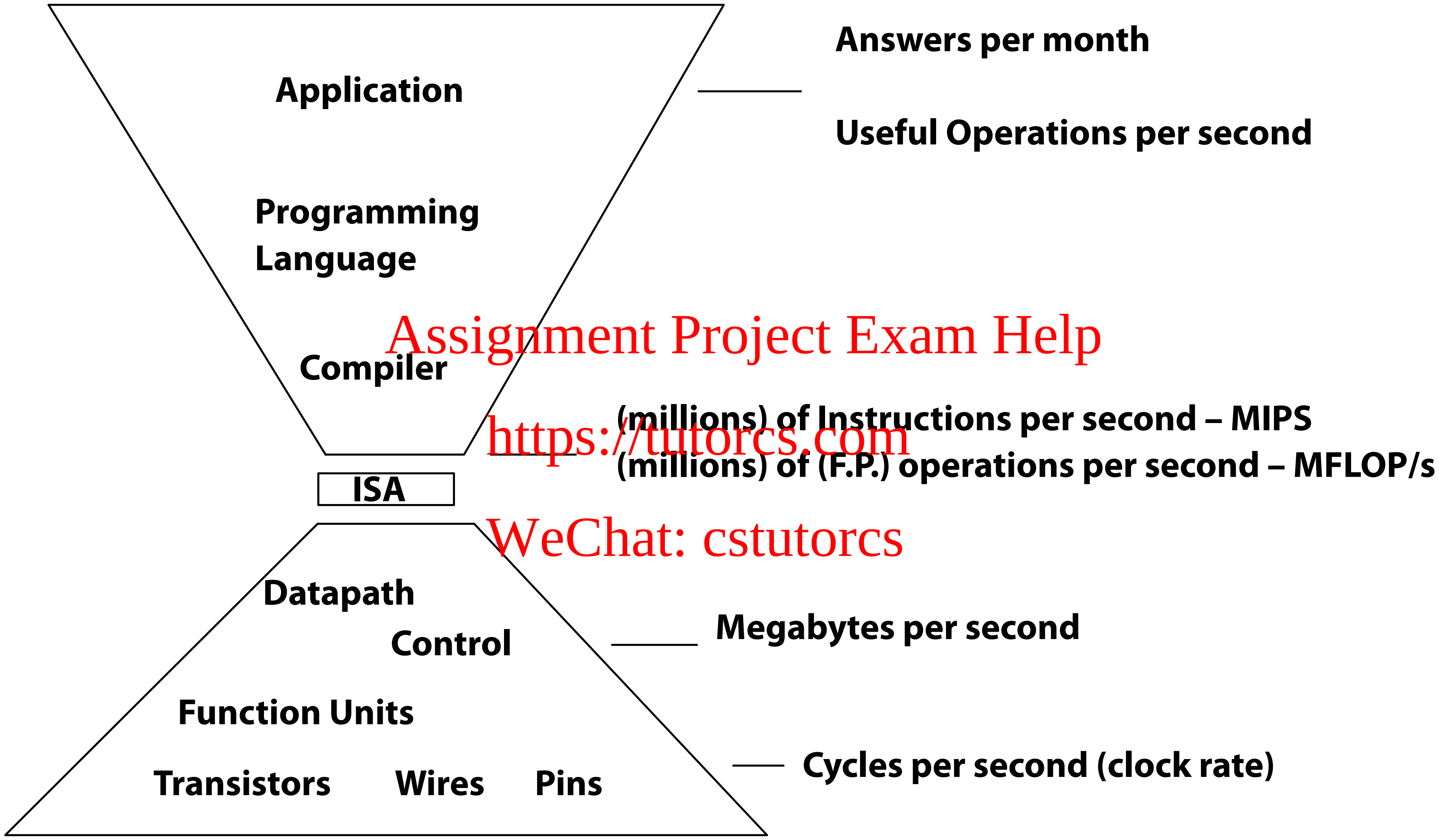
# Today's Goals

- Understand performance, speedup, throughput, latency
- Relationship between cycle time, cycles/instruction (CPI), number of instructions (the *performance equation*)
- Amdahl's Law
- Benchmarks
- Know how to do problems at end of lecture!

# Performance

- Measure, Report, and Summarize
- Make intelligent choices
- See through the marketing hype
- Key to understanding underlying organizational motivation
  - 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?)
  - How does the machine's instruction set affect performance?

# Metrics of performance



Application

Programming Language

Compiler

ISA

Datapath

Control

Function Units

Transistors Wires Pins

Answers per month

Useful Operations per second

(millions) of Instructions per second – MIPS

(millions) of (F.P.) operations per second – MFLOP/s

Megabytes per second

Cycles per second (clock rate)

- Each metric has a place and a purpose, and each can be misused

# Definitions

- Performance is in units of things-per-time
  - Miles per hour, bits per second, widgets per day …
  - Bigger is better
- If we are primarily concerned with response time:
  - Performance(x) = 1 / ExecutionTime(x)
- "X is n times faster than Y" means
  - n = Performance(X) / Performance(Y) = Speedup
  - If X is 1.yz times faster than Y, we can informally say that X is yz% faster than Y. Speedup is better.

# Latency vs. Throughput

- *Latency* (Response Time)
  - 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?
- 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?

# Two notions of "performance"

| Plane | DC to Paris | Speed | Passengers | Throughput (pmph) |
|---|---|---|---|---|
| Boeing 747 | 6.5 hours | 610 mph | 470 | 286,700 |
| Concorde | 3 hours | 1350 mph | 132 | 178,200 |

- Which has higher performance?
- Time to do the task (Execution Time)
  - execution time, response time, latency
- Tasks per day, hour, week, sec, ns … (Performance)
  - throughput, bandwidth
- Response time and throughput often are in opposition

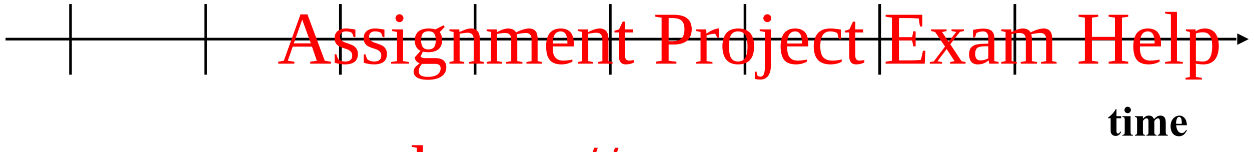
# Example

- Time of Concorde vs. Boeing 747?
    - Concorde is 1350 mph / 610 mph = 2.2 times faster
    - = 6.5 hours / 3 hours
- Throughput of Concorde vs. Boeing 747 ?
    - Concorde is 178,200 pmph / 286,700 pmph = 0.62 “times faster”
    - Boeing is 286,700 pmph / 178,200 pmph = 1.60 “times faster”
- Boeing is 1.6 times (“60%”) faster in terms of throughput
- Concorde is 2.2 times (“120%”) faster in terms of flying time
- We will focus primarily on execution time for a single job
    - But sysadmins (or server-based companies) may use throughput as their primary metric!

# 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
- Cycle time = time between ticks = seconds per cycle
- Clock rate (frequency) = cycles per second (1 Hz = 1 cycle/sec)
    - A 200 MHz clock has a cycle time of . . .

$$\frac{1}{200 \times 10^{6}} \times 10^{9} = 5 \text{ nanoseconds}$$

# Clock Speed Is Not The Whole Story

| | *SPECint95* | *SPECfp95* |
|---|---|---|
| 195 MHz MIPS R10000 | 11.0 | 17.0 |
| 400 MHz Alpha 21164 | 12.3 | 17.2 |
| 300 MHz UltraSPARC | 12.1 | 15.5 |
| 300 MHz Pentium II | 11.6 | 8.8 |
| 300 MHz PowerPC G3 | 14.8 | 11.4 |
| 135 MHz POWER2 | 6.2 | 17.6 |

[http://www.pattosoft.com.au/Articles/ModernMicroprocessors/]

# How to Improve Performance

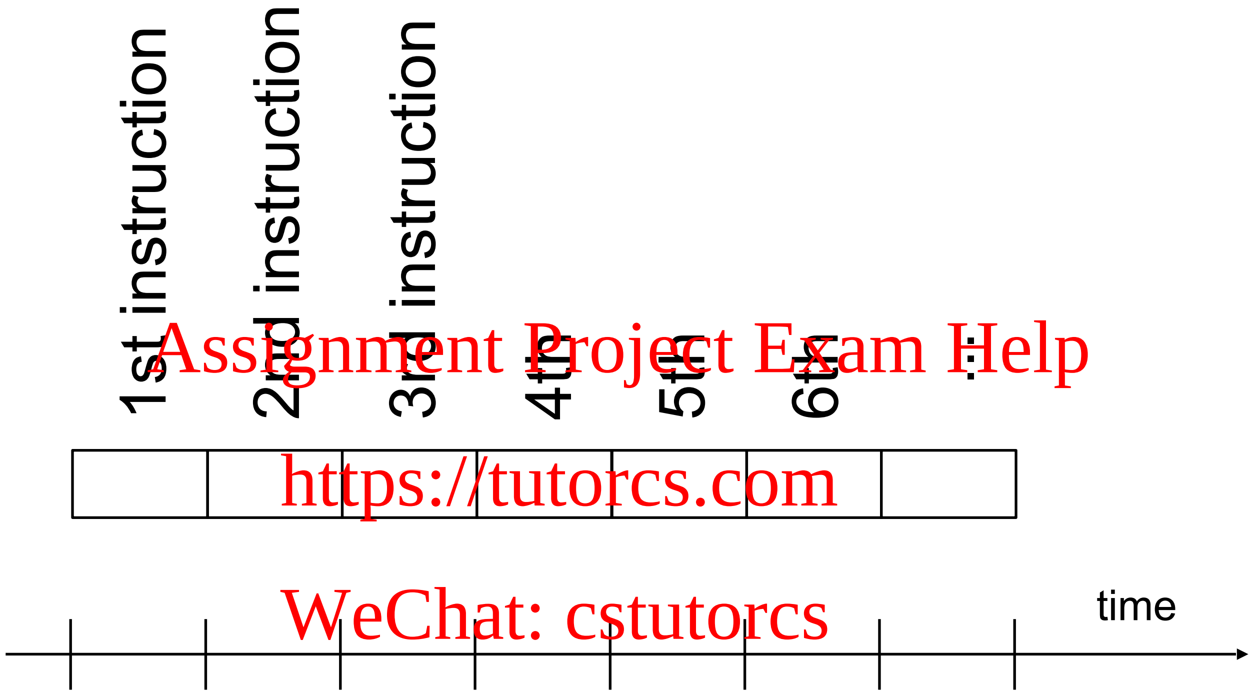
- So, to improve performance (everything else being equal) you can either (increase/ decrease):

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

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

# How many cycles in a program?

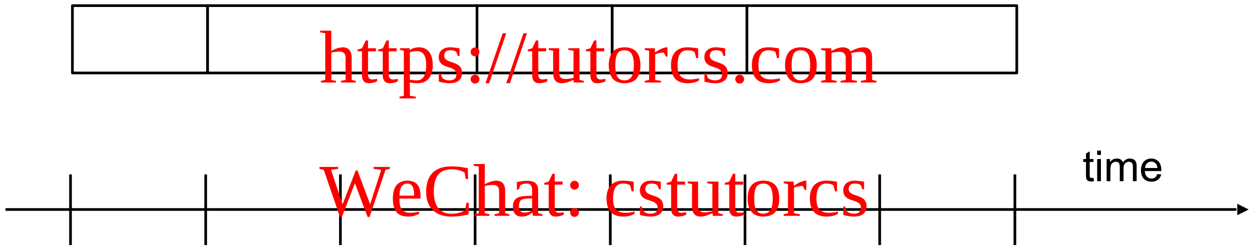
- Could assume that # of cycles = # of instructions



- This assumption is incorrect:
  - different instructions take different amounts of time on different machines (even with the same instruction set).
  - Why?

# Different #s of cycles for diff'nt instrs

- Multiplication takes more time than addition
- Floating point operations take a different amount of time (generally) 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 (more later)

# Example instruction latencies

Imagine Stream Processor:

On ALU:

- Integer adds: 2 cycles
- FP adds: 4 cycles
- Logic ops (and, or, xor): 1
- Equality: 1
- < or >: 2
- Shifts: 1
- Float->int: 3
- Int->float: 4
- Select (a?b:c): 1

Other functional units:

- Integer multiply: 4
- Integer divide: 22
- Integer remainder: 23
- FP multiply: 4
- FP divide: 17
- FP sqrt: 16

# CPI

- How many clock cycles, *on average*, does it take for every instruction executed?
- We call this CPI ("Cycles Per Instruction").
- Its inverse (1/CPI) is IPC ("Instructions Per Cycle").

- CISC machines: this number (CPI) is … higher/lower?
- RISC machines: this number (CPI) is … higher/lower?

# CPI: Average Cycles per Instruction

- CPI = (CPU Time * Clock Rate) / Instruction Count
  - = Clock Cycles / Instruction Count

$$\text{CPI} = \sum_{i=1}^{n} \text{CPI}_i \times F_i \quad \text{where } F_i = \frac{I_i}{\text{Instruction Count}}$$

- On Imagine, integer adds are 2 cycles, FP adds are 4.
- Consider an application that has 1/3 integer adds and 2/3 FP adds.
- What is its CPI?

- Given a 3 GHz machine, how many instrs/sec?

# Dimensional Analysis

- Given clock speed and CPI, how many instrs/sec?

- We have:
  cycles/second, cycles/instruction

*This is a useful debugging tool!*

- We want:
  instructions/second

- So:

$$\frac{\text{instructions}}{\text{cycle}} \times \frac{\text{cycles}}{\text{second}} = \frac{\text{instructions}}{\text{second}}$$

# The Performance Equation

- Time = Cycle Time * CPI * Instruction Count
  - = seconds/cycle * cycles/instr * instrs/program
  - => seconds/program
- Performance = Clock Rate * IPC * 1/I
  - = cycles/second * instr/cycle * program/instr
  - => programs/second
    - Clock rate * IPC = instr/second = IPS (MIPS)
- "The only reliable measure of computer performance is time."

# 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)

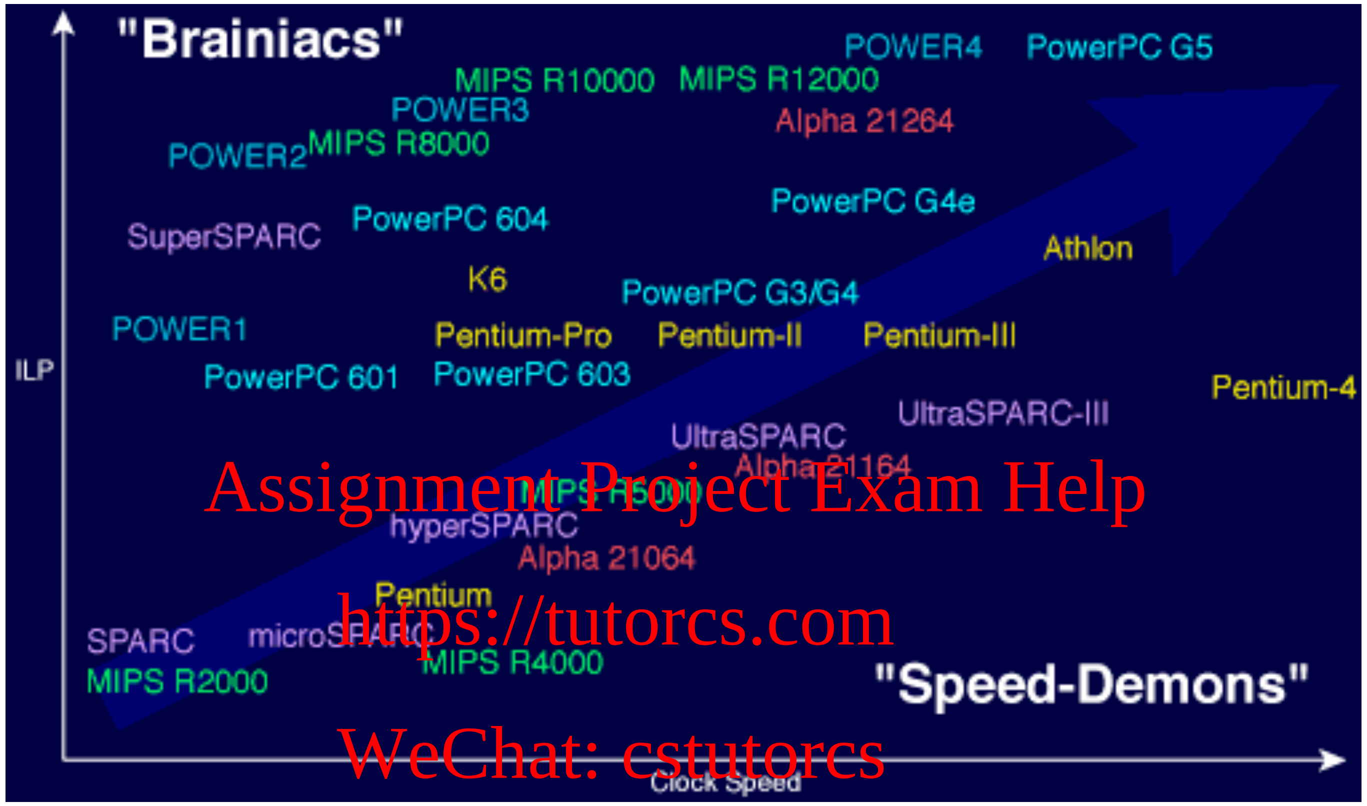
    this would be higher for a program using simple instructions

# 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?

- Common pitfall: thinking one of the variables is indicative of performance when it really isn't.

# Brainiacs vs. Speed Demons



- Modern processor design balances CPI and clock speed
  - Brainiacs do more work per clock cycle
  - Speed demons have faster clock cycles

*[http://www.pattosoft.com.au/ Articles/ModernMicroprocessors/]*

# AMD “True Performance Initiative”

- AMD Unveils New AMD Athlon™ XP Processor; Drives Initiative to Develop New Processor Performance Metric
- SAN FRANCISCO, CA -- October 9, 2001 -- AMD (NYSE: AMD) today announced the new AMD Athlon™ XP processor, the world’s highest-performance processor for desktop PCs. AMD also announced plans to drive an initiative to develop a *reliable processor performance metric that PC users can trust*. The True Performance Initiative reflects AMD’s continued commitment to business and home PC users. … AMD will identify the AMD Athlon XP processor using model numbers, as opposed to clock speed in megahertz, and is introducing 1800+, 1700+, 1600+ and 1500+ versions.

# AMD "True Performance Initiative"

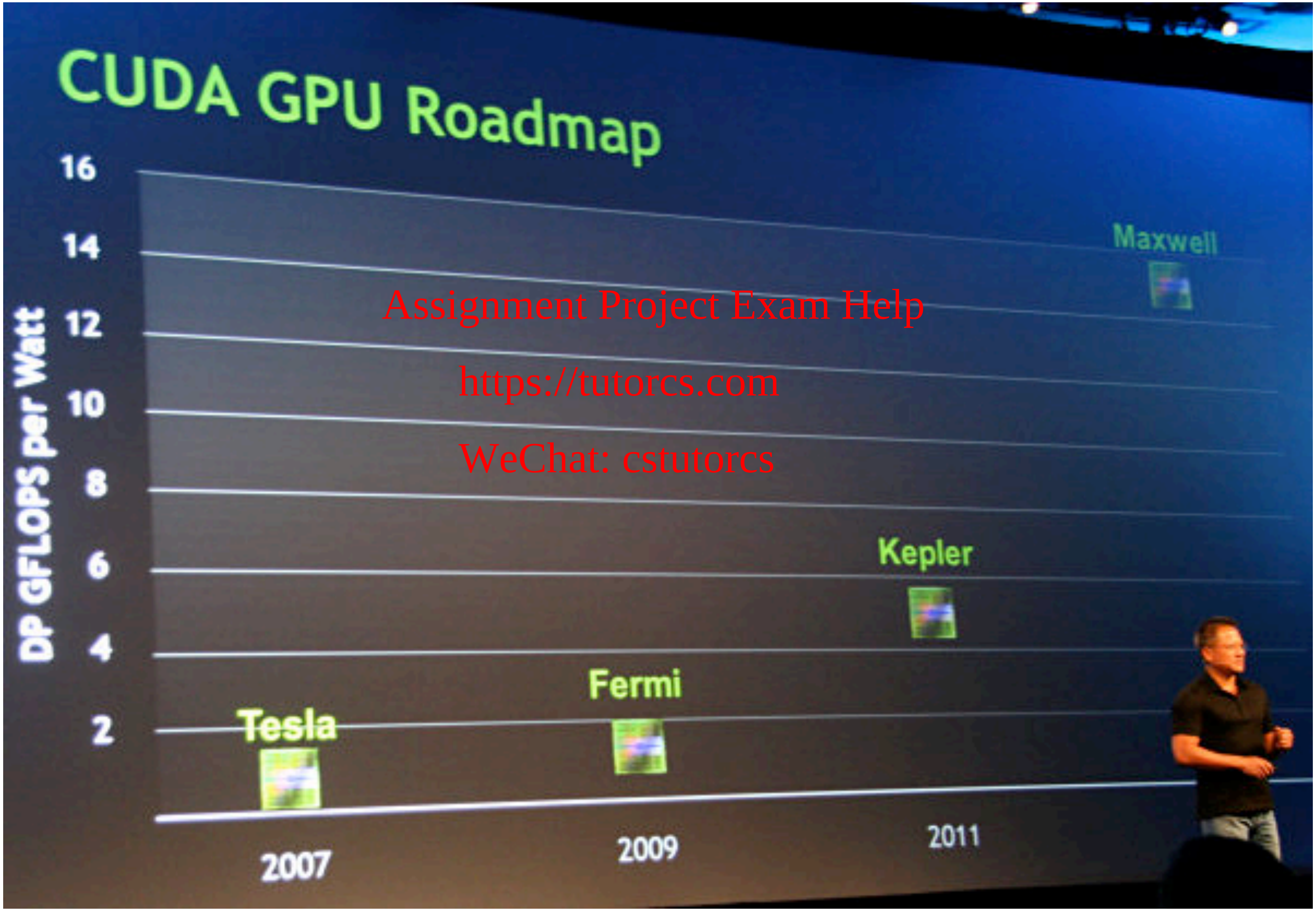
- "For most of the PC's first 20 years, megahertz was a reliable indicator of PC processor performance because the major players used the same architecture for product design, and clock speed was a good proxy for performance. This is no longer true. The award-winning performance of our seventh-generation AMD Athlon processor architecture demonstrates that clock speed is only half of the performance equation." [W.J. Sanders III, AMD founder]

*[http://www.amd.com/us-en/Corporate/InvestorRelations/0,,51_306_2317_6363,00.html]*

# Intel: Performance Per Watt

- "Intel Swaps Clock Speed for Power Efficiency"
  John Spooner, 15 August 2005, eweek.com
- "Intel, which next week is expected to announce plans to move to a new processor architecture, is switching to a new yardstick to measure processor performance: performance per watt. … Intel's announcement will publicly signal an internal shift that's already taken place. After years of promoting clock speed, it's now emphasizing overall performance and power-efficiency."

# NVIDIA GTC Keynote September 2010



# Aspects of CPU Performance

$$\text{CPU time} = \frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}$$

| | Instr count | CPI | Clock rate |
|---|---|---|---|
| Program | | | |
| Compiler | | | |
| Instruction Set | | | |
| Organization | | | |
| Technology | | | |

# Remember

- Performance is specific to a particular program/s
  - Total execution time is a consistent summary of performance

- For a given architecture performance increases come from:
  - increases in clock rate (without adverse CPI affects)
  - improvements in processor organization that lower CPI
  - compiler enhancements that lower CPI and/or instruction count

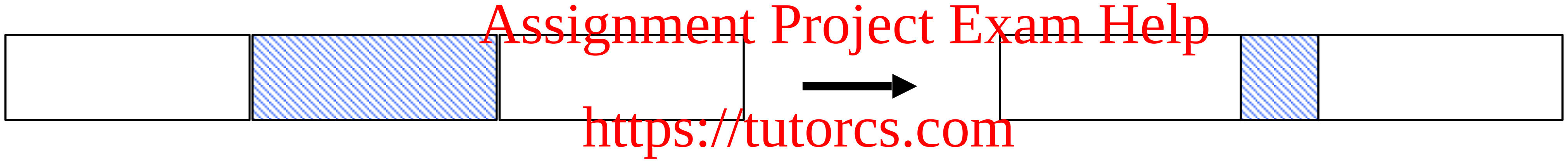
- Pitfall: expecting improvement in one aspect of a machine's performance to affect the total performance

- You should not always believe everything you read! Read carefully! (see newspaper articles)

# Amdahl's Law

Speedup due to enhancement E:

$$\text{Speedup(E)} = \frac{\text{ExTime w/o E}}{\text{ExTime w/ E}} = \frac{\text{Performance w/ E}}{\text{Performance w/o E}}$$



Suppose that enhancement E accelerates a fraction F of the task by a factor S and the remainder of the task is unaffected:

$$\text{ExTime (with E)} = ((1\text{-}F) + F/S) * \text{ExTime(without E)}$$

$$\text{Speedup (with E)} = \frac{1}{(1\text{-}F) + F/S}$$

Design Principle: ***Make the common case fast!***

There are many ways to express Amdahl's Law!

# Undergrad Productivity

- Average ECE student spends:
- 4 hours sleeping
- 2 hours eating
- 18 hours studying
- Magic pill gives you all sleeping, eating in 1 minute!
- What’s the speedup on sleeping/eating?
- How much more productive can you get?

# Undergrad Productivity

$$\text{Speedup (with E)} = \frac{1}{(1-F) + F/S}$$

F = accelerated fraction = 0.25 (6 hrs/24 hrs)

S = speedup = 6 hrs / 1 minute = 360

Overall speedup:

- 1 / [(1-0.25) + (0.25/360)]
- ~= 1 / (1-0.25)
- ~= 1.33
- 33% more productive!

# Benchmarks

- Performance best determined by running a real application
  - Use programs typical of expected workload
  - Or, typical of expected class of applications (e.g., compilers/editors, scientific applications, graphics, etc.)
- Small benchmarks
  - nice for architects and designers
  - easy to standardize
  - can be abused
- SPEC (System Performance Evaluation Cooperative)
  - companies have agreed on a set of real program and inputs
  - can still be abused
  - valuable indicator of performance (and compiler technology)

# Discussion Section

# 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 10 ns and a CPI of 2.0
  - Machine B has a clock cycle time of 20 ns and a CPI of 1.2
  - What machine is faster for this program, and by how much?
- If two machines have the same ISA for a given program which of our quantities (e.g., clock rate, CPI, execution time, # of instructions, MIPS) will always be identical?

# # of 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
  - 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 3 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.

# MIPS example

- Which sequence will be faster according to MIPS?

- Which sequence will be faster according to execution time?

# Example (RISC processor)

- Base Machine (Reg / Reg)

| Op | Freq | Cycles | CPI(i) | % Time |
|---|---|---|---|---|
| ALU | 50% | 1 | | |
| Load | 20% | 5 | | |
| Store | 10% | 3 | | |
| Branch | 20% | 2 | | |

Typical Mix

- What’s the CPI?

# Example (RISC processor)

- Base Machine (Reg / Reg)

| Op | Freq | Cycles | CPI(i) | % Time |
|---|---|---|---|---|
| ALU | 50% | 1 | 0.5 | 23% |
| Load | 20% | 5 | 1.0 | 45% |
| Store | 10% | 3 | 0.3 | 14% |
| Branch | 20% | 2 | 0.4 | 18% |

Typical Mix

- What's the CPI?

  0.5 + 1.0 + 0.3 + 0.4 = 2.2

# Example (RISC processor)

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

# Example (RISC processor)

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

# Example (Amdahl's Law 1)

- **Execution Time After Improvement =**
  **Execution Time Unaffected + (Execution Time Affected / Amount of Improvement)**

- **Example:**
  **"Suppose a program runs in 100 seconds on a machine, with multiply responsible for 80 seconds of this time. How much do we have to improve the speed of multiplication if we want the program to run 4 times faster?"**

  **How about making it 5 times faster?**

# Example (Amdahl's Law 2)

- Suppose we enhance a machine making all floating-point instructions run five times faster. If the execution time of some benchmark before the floating-point enhancement is 10 seconds, what will the speedup be if half of the 10 seconds is spent executing floating-point instructions?

# Example (Amdahl's Law 3)

- We are looking for a benchmark to show off the new floating-point unit described in Part 2, and want the overall benchmark to show a speedup of 3. We are considering a benchmark that runs for 100 seconds with the old floating-point hardware. How much of the execution time would floating-point instructions have to account for in this program in order to yield our desired speedup on this benchmark?

# Example (Compiler Optimization)

- You want to understand the performance of a specific program on your 3.3 GHz machine. You collect the following statistics for the instruction mix and breakdown:

| Instruction Class | Frequency (%) | Cycles |
|---|---|---|
| Arithmetic/logical | 50 | 1 |
| Load | 20 | 2 |
| Store | 10 | 2 |
| Jump | 10 | 1 |
| Branch | 10 | 3 |

# Part A

- Calculate the CPI and MIPS for this program.

# Part B

- Your compiler team reports they can eliminate 20% of ALU instructions (i.e. 10% of all instructions). What is the speedup?

# Part C

- With the compiler improvements, what is the new CPI and MIPS?

# My notes

1/2 instr/cycle * 10^8 cycles/sec = 50M instr/sec
1/1.2 instr/cycle * 5*10^7 cycles/sec = 42M instr/sec
A is faster by 50/42 ~ 20%
Same ISA: only number of instructions will be identical
==
Seq 1: 2*1 + 1*2 + 2*3 / 5 total = 10/5 = 2 = CPI     10 cycles
Seq 2: 4*1 + 1*2 + 1*3 / 6 total = 9/6 = 1.5 = CPI   9 cycles: 11% faster
==
MIPS: Compiler 1: 5M*1 + 1M*2 + 1M*3 / 7M = 10/7 CPI = 1.42   IPC = 0.7
Compiler 2: 10M*1 + 1M*2 + 1M*3 / 12M = 15/12 = 1.25   IPC = 0.8

MIPS: Assuming same computer, one with highest IPC (#2) 0.8 vs 0.7
Time: Assuming same computer, one with smallest cycles/program = cycles/instr*instr/program
#2: 1.25 * 12M = 15 MC/program
#1: 1.42 * 7M = 10MC/program, #1 is faster
==
CPI = 0.5*1 + 0.2*5 + 0.1*3 + 0.2*2 = 2.2
==
Load time down to 2 cycles: LD goes from 5*0.2 to 2*0.2
CPI = 0.5*1 + 0.2*2 + 0.1*3 + 0.2*2 = 1.6
Speedup = 2.2/1.6 = 1.375
==
Branch -> 1:
CPI = 0.5*1 + 0.2*5 + 0.1*3 + 0.2*1 = 2.0
New CPI = 2; speedup = 2.2/2.0 = 10%
==
ALU now takes 0.5 ops instead of one
CPI = 0.5*0.5 + 0.2*5 + 0.1*3 + 0.2*2 = 1.95
speedup=2.2/1.95 = 13%
Deleting half: changes instruction mix
Mix now ALU (33.3), load (26.7), store (13.3), branch (26.7)
CPI now 0.33*1 + 0.267*5 + 0.133*3 + 0.267*2 =  2.6
But number of instructions is now 0.75 of what it was before
CPI_new = cycles_new/instructions_new = cycles_new/instruction_old * instruction_old/instruction_new = 2.6 * 0.75 = 1.95
Same speedup

Doesn’t always work out that way.
==
25 = 20 + 80/x   x = 16x faster
20 = 20 + 80/x  x = infinitely fast
==
Time after = unaffected + affected/speedup
? = 5 + 5/5 => 6 seconds
Speedup = 10/6 = 1.67
==
Speedup = 100 seconds / 3 = 33.3 seconds
33.3 = (100-x) + x/5
= 100 - 4/5 x
4/5 x = 200/3
X = 200/3 * 5/4 = 1000/12 = 83%
==
CPI = 0.5*1 + 0.2*2 + 0.1*2 + 0.1*1 + 0.1*3 = 1.5
MIPS = IPC * clock speed = 3.3 GHz/1.5 = 2200 MIPS
==
Long way:
4/9*1 + 2/9*2 + 1/9*2 + 1/9*1 + 1/9*3 = 14/9 = 1.56
Orig: cycles/prog = CPI * instr/prog = 1.5*I
New: 14/9 * 0.9I = 1.4I     cycles / instr_new * instr_new
Speedup = 1.5 / 1.4 = 1.07
Short way:
Cycles per equiv of avg old instruction (cycles_new / instr_old * instr_old)
0.4*1 + 0.2*2 + 0.1*2 + 0.1*1 + 0.1*3 = 1.4
==
CPI = 1.56
MIPS = IPC * clock speed = 3.3GHz / 1.56 = 21.2 MIPS