
CPE 631 Lecture 02: Fundamentals of Computer Design (part 2)

Electrical and Computer Engineering
University of Alabama in Huntsville
Aleksandar Milenkovic, milenka@ece.uah.edu
<http://www.ece.uah.edu/~milenka>

CPE
631
O AM

Before we start

“What is man in nature?
Nothing in relation to the infinite,
everything in relation to nothing,
a mean between nothing and everything.”

Blaise Pascal, 1670

05/01/2004

UAH-CPE631

2

CPE
631
O AM

Outline

- Review
- Measuring and Reporting Performance
- Quantitative Principles of Computer Design
- Things to Remember

05/01/2004

UAH-CPE631

3

CPE
631
O AM

Review

- Computing classes: desktop, server, embedd.
- Technology trends

	Capacity	Speed
Logic	4x in 3+ years	2x in 3 years
DRAM	4x in 3-4 years	33% in 10 years
Disk	4x in 3-4 years	33% in 10 years

- Cost

- Learning curve:
manufacturing costs decrease over time
- Volume: the number of chips manufactured
- Commodity

05/01/2004

CPE 631

4

Review

■ Cost of an integrated circuit

$$IC\ cost = \frac{Die\ cost + Testing\ cost + Packaging\ cost}{Final\ test\ yield}$$

$$Cost\ of\ die = \frac{Cost\ of\ wafer}{Dies\ per\ wafer \cdot Die\ yield}$$

$$Dies\ per\ wafer = \frac{p \cdot (Wafer\ diameter / 2)^2}{Die\ area} = \frac{p \cdot Wafer\ diameter}{\sqrt{2} \cdot Die\ area}$$

$$Die\ yield = Wafer\ yield \cdot \frac{1}{1 + \frac{Defects\ per\ unit\ area \cdot Die\ area}{a}}$$

05/01/2004

CPE 631

5

Cost-Performance

- Purchasing perspective: from a collection of machines, choose one which has
 - best performance?
 - least cost?
 - best performance/cost?
- Computer designer perspective: faced with design options, select one which has
 - best performance improvement?
 - least cost?
 - best performance/cost?
- Both require: basis for comparison and metric for evaluation

05/01/2004

UAH-CPE631

6

Two “notions” of performance

- Which computer has better performance?
 - User: one which runs a program in less time
 - Computer centre manager: one which completes more jobs in a given time
- Users are interested in reducing Response time or Execution time
 - the time between the start and the completion of an event
- Managers are interested in increasing Throughput or Bandwidth
 - total amount of work done in a given time

05/01/2004

UAH-CPE631

7

An Example

Plane	DC to Paris [hour]	Top Speed [mph]	Passengers	Throughput [p/h]
Boeing 747	6.5	610	470	72
Concorde	3	1350	132	44 (=132/3)

- Which has higher performance?
 - Time to deliver 1 passenger?
 - Concorde is $6.5/3 = 2.2$ times faster (120%)
 - Time to deliver 400 passengers?
 - Boeing is $72/44 = 1.6$ times faster (60%)

05/01/2004

UAH-CPE631

8

Definition of Performance

- We are primarily concerned with Response Time

- Performance [things/sec]

$$Performance(x) = \frac{1}{Execution_time(x)}$$

- "X is n times faster than Y"

$$n = \frac{Execution_time(y)}{Execution_time(x)} = \frac{Performance(x)}{Performance(y)}$$

- As faster means both increased performance and decreased execution time, to reduce confusion will use "improve performance" or "improve execution time"

05/01/2004

UAH-CPE631

9

Execution Time and Its Components

- Wall-clock time, response time, elapsed time
 - the latency to complete a task, including disk accesses, memory accesses, input/output activities, operating system overhead,...

- CPU time

- the time the CPU is computing, excluding I/O or running other programs with multiprogramming
- often further divided into user and system CPU times

- User CPU time

- the CPU time spent in the program

- System CPU time

- the CPU time spent in the operating system

05/01/2004

UAH-CPE631

10

UNIX time command

90.7u 12.9s 2:39 65%

- 90.7 - seconds of user CPU time
- 12.9 - seconds of system CPU time
- 2:39 - elapsed time (159 seconds)
- 65% - percentage of elapsed time that is CPU time
(90.7 + 12.9)/159

05/01/2004

UAH-CPE631

11

CPU Execution Time

CPU time = CPU clock cycles for a program / Clock cycle time

$$CPUtime = \frac{CPU\ clock\ cycles\ for\ a\ program}{Clock\ rate}$$

Definitions

- Instruction count (IC) = Number of instructions executed

- Clock cycles per instruction (CPI)

$$CPI = \frac{CPU\ clock\ cycles\ for\ a\ program}{IC}$$

CPI - one way to compare two machines with same instruction set, since Instruction Count would be the same

05/01/2004

UAH-CPE631

12

CPU Execution Time (cont'd)

$$CPU\ time = IC \cdot CPI \cdot Clock\ cycle\ time$$

$$CPU\ time = \frac{IC \cdot CPI}{Clock\ rate}$$

$$CPU\ time = \frac{Instructions}{Program} \cdot \frac{Clock\ cycles}{Instruction} \cdot \frac{Seconds}{Clock\ cycle} = \frac{Seconds}{Program}$$

	IC	CPI	Clock rate
Program	X		
Compiler	X	(X)	
ISA	X	X	
Organisation		X	X
Technology			X

05/01/2004

UAH-CPE631

13

How to Calculate 3 Components?

- **Clock Cycle Time**
 - in specification of computer (Clock Rate in advertisements)
- **Instruction count**
 - Count instructions in loop of small program
 - Use simulator to count instructions
 - Hardware counter in special register (Pentium II)
- **CPI**
 - Calculate: Execution Time / Clock cycle time / Instruction Count
 - Hardware counter in special register (Pentium II)

05/01/2004

UAH-CPE631

14

Another Way to Calculate CPI

- First calculate CPI for each individual instruction (add, sub, and, etc.): CPI_i
- Next calculate frequency of each individual instr.: $Freq_i = IC_i/IC$
- Finally multiply these two for each instruction and add them up to get final CPI

$$CPI = \sum_{i=1}^n \frac{IC_i}{IC} \times CPI_i$$

Op	Freq _i	CPI _i	Prod.	% Time
ALU	50%	1	0.5	23%
Load	20%	5	1.0	45%
Store	10%	3	0.3	14%
Bran.	20%	2	0.4	18%

Z.Z

05/01/2004

UAH-CPE631

15

Choosing Programs to Evaluate Per.

- Ideally run typical programs with typical input before purchase, or before even build machine
 - Engineer uses compiler, spreadsheet
 - Author uses word processor, drawing program, compression software
- **Workload** – mixture of programs and OS commands that users run on a machine
- Few can do this
 - Don't have access to machine to "benchmark" before purchase
 - Don't know workload in future

05/01/2004

UAH-CPE631

16

Benchmarks

- Different types of benchmarks
 - Real programs (Ex. MSWord, Excel, Photoshop,...)
 - Kernels - small pieces from real programs (Linpack,...)
 - Toy Benchmarks - short, easy to type and run (Sieve of Erathosthenes, Quicksort, Puzzle,...)
 - Synthetic benchmarks - code that matches frequency of key instructions and operations to real programs (Whetstone, Dhrystone)
- Need industry standards so that different processors can be fairly compared
- Companies exist that create these benchmarks: "typical" code used to evaluate systems

Benchmark Suites

- SPEC - Standard Performance Evaluation Corporation (www.spec.org)
 - originally focusing on CPU performance SPEC89|92|95, SPEC CPU2000 (11 Int + 13 FP)
 - graphics benchmarks: SPECviewperf, SPECcapc
 - server benchmark: SPECSFS, SPECWEB
- PC benchmarks (Winbench 99, Business Winstone 99, High-end Winstone 99, CC Winstone 99) (www.zdnet.com/etestinglabs/filters/benchmarks)
- Transaction processing benchmarks (www.tpc.org)
- Embedded benchmarks (www.eembc.org)

Comparing and Summarising Per.

An Example

Program	Com. A	Com. B	Com. C
P1 (sec)	1	10	20
P2 (sec)	1000	100	20
Total (sec)	1001	110	40

- A is 20 times faster than C for program P1
- C is 50 times faster than A for program P2
- B is 2 times faster than C for program P1
- C is 5 times faster than B for program P2

- What we can learn from these statements?
- We know nothing about relative performance of computers A, B, C!
- One approach to summarise relative performance: use total execution times of programs

Comparing and Sum. Per. (cont'd)

- Arithmetic mean (AM) or weighted AM to track time

$$\frac{1}{n} \sum_{i=0}^n Time_i \quad \text{Time}_i - \text{execution time for } i\text{th program}$$

$$w_i - \text{frequency of that program in workload}$$

- Harmonic mean or weighted harmonic mean of rates tracks execution time

$$\frac{n}{\sum_{i=0}^n \frac{1}{Rate_i}}, \quad Rate_i = \frac{1}{Time_i} \quad \frac{1}{\sum_{i=0}^n \frac{w_i}{Rate_i}}$$

- Normalized execution time to a reference machine

- do not take arithmetic mean of normalized execution times, use geometric mean

Problem: GM rewards equally the following improvements:
 Program A: from 2s to 1s, and
 Program B: from 2000s to 1000s

Quantitative Principles of Design

- Where to spend time making improvements?
⇒ **Make the Common Case Fast**

- Most important principle of computer design:
Spend your time on improvements where those improvements will do the most good

- Example

- Instruction A represents 5% of execution
- Instruction B represents 20% of execution
- Even if you can drive the time for A to 0, the CPU will only be 5% faster

- Key questions

- What the frequent case is?
- How much performance can be improved by making that case faster?

05/01/2004 UAH-CPE631

21

Amdahl's Law

- Suppose that we make an enhancement to a machine that will improve its performance;
Speedup is ratio:

$$\text{Speedup} = \frac{\text{ExTime for entire task without enhancement } t}{\text{ExTime for entire task using enhancement } t}$$

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

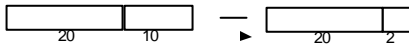
- Amdahl's Law states that the performance improvement that can be gained by a particular enhancement is limited by the amount of time that enhancement can be used

05/01/2004

UAH-CPE631

22

Computing Speedup



- Fraction_{enhanced} = fraction of execution time in the original machine that can be converted to take advantage of enhancement (E.g., 10/30)
- Speedup_{enhanced} = how much faster the enhanced code will run (E.g., 10/2=5)
- Execution time of enhanced program will be sum of old execution time of the unenhanced part of program and new execution time of the enhanced part of program:

$$\text{ExTime}_{\text{new}} = \text{ExTime}_{\text{unenhanced}} + \frac{\text{ExTime}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}$$

05/01/2004

UAH-CPE631

23

Computing Speedup (cont'd)

- Enhanced part of program is Fraction_{enhanced} so times are:

$$\text{ExTime}_{\text{unenhanced}} = \text{ExTime}_{\text{old}} \cdot (1 - \text{Fraction}_{\text{enhanced}})$$

$$\text{ExTime}_{\text{enhanced}} = \text{ExTime}_{\text{old}} \cdot \text{Fraction}_{\text{enhanced}}$$

- Factor out Time_{old} and divide by Speedup_{enhanced}:

$$\text{ExTime}_{\text{new}} = \text{ExTime}_{\text{old}} \cdot \left(1 - \text{Fraction}_{\text{enhanced}} + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}} \right)$$

- Overall speedup is ratio of Time_{old} to Time_{new}:

$$\text{Speedup} = \frac{1}{1 - \text{Fraction}_{\text{enhanced}} + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}}$$

05/01/2004

UAH-CPE631

24

An Example

- Enhancement runs 10 times faster and it affects 40% of the execution time
- $Fraction_{enhanced} = 0.40$
- $Speedup_{enhanced} = 10$
- $Speedup_{overall} = ?$

$$Speedup = \frac{1}{1 - 0.4 + \frac{0.4}{10}} = \frac{1}{0.64} \approx 1.56$$

"Law of Diminishing Returns"

- Suppose that same piece of code can now be enhanced another 10 times
- $Fraction_{enhanced} = 0.04/(0.60 + 0.04) = 0.0625$
- $Speedup_{enhanced} = 10$

$$Speedup_{overall} = \frac{1}{1 - Fraction_{enhanced} + \frac{Fraction_{enhanced}}{Speedup_{enhanced}}}$$

$$Speedup_{overall} = \frac{1}{0.94 + \frac{0.06}{10}} \approx 1.059$$

Using CPU Performance Equations

- Example #1: consider 2 alternatives for conditional branch instructions
 - CPU A: a condition code (CC) is set by a compare instruction and followed by a branch instruction that test CC
 - CPU B: a compare is included in the branch
 - Assumptions:
 - on both CPUs, the conditional branch takes 2 clock cycles
 - all other instructions take 1 clock cycle
 - on CPU A, 20% of all instructions executed are cond. branches; since every branch needs a compare, another 20% are compares
 - because CPU A does not have a compare included in the branch, assume its clock cycle time is 1.25 times faster than that of CPU B
 - Which CPU is faster?
 - Answer the question when CPU A clock cycle time is only 1.1 times faster than that of CPU B

Using CPU Performance Eq. (cont'd)

- Example #1 Solution:
 - CPU A
 - $CPI(A) = 0.2 \times 2 + 0.8 \times 1 = 1.2$
 - $CPU_time(A) = IC(A) \times CPI(A) \times Clock_cycle_time(A)$
 $= IC(A) \times 1.2 \times Clock_cycle_time(A)$
 - CPU B
 - $CPU_time(B) = IC(B) \times CPI(B) \times Clock_cycle_time(B)$
 - $Clock_cycle_time(B) = 1.25 \times Clock_cycle_time(A)$
 - $IC(B) = 0.8 \times IC(A)$
 - $CPI(B) = ?$ compares are not executed in CPU B, so 20%/80%, or 25% of the instructions are now branches
 $CPI(B) = 0.25 \times 2 + 0.75 \times 1 = 1.25$
 - $CPU_time(B) = 0.8 \times IC(A) \times 1.25 \times 1.25 \times Clock_cycle_time(A)$
 $= 1.25 \times IC(A) \times Clock_cycle_time(A)$
 - $CPU_time(B)/CPU_time(A) = 1.25/1.2 = 1.04167 \Rightarrow$
 CPU A is faster for 4.2%

CPE 631 0 AM

MIPS as a Measure for Comparing Performance among Computers

- MIPS – Million Instructions Per Second

$$MIPS = \frac{IC}{CPU\ time \cdot 10^6}$$

$$CPU\ time = \frac{IC \cdot CPI}{Clock\ rate}$$

$$MIPS = \frac{IC}{\frac{IC \cdot CPI}{Clock\ rate} \cdot 10^6} = \frac{Clock\ rate}{CPI \cdot 10^6}$$

05/01/2004 UAH-CPE631 29

CPE 631 0 AM

MIPS as a Measure for Comparing Performance among Computers (cont'd)

- Problems with using MIPS as a measure for comparison
 - MIPS is dependent on the instruction set, making it difficult to compare MIPS of computers with different instruction sets
 - MIPS varies between programs on the same computer
 - Most importantly, MIPS can vary inversely to performance
 - Example: MIPS rating of a machine with optional FP hardware
 - Example: Code optimization

05/01/2004 UAH-CPE631 30

CPE 631 0 AM

MIPS as a Measure for Comparing Performance among Computers (cont'd)

- Assume we are building optimizing compiler for the load-store machine with following measurements

Ins. Type	Freq.	Clock cycle count
ALU ops	43%	1
Loads	21%	2
Stores	12%	2
Branches	24%	2

- Compiler discards 50% of ALU ops
- Clock rate: 500MHz
- Find the MIPS rating for optimized vs. unoptimized code? Discuss it.

05/01/2004 UAH-CPE631 31

CPE 631 0 AM

MIPS as a Measure for Comparing Performance among Computers (cont'd)

- Unoptimized
 - $CPI(u) = 0.43 \times 1 + 0.57 \times 2 = 1.57$
 - $MIPS(u) = 500MHz / (1.57 \times 10^6) = 318.5$
 - $CPU_time(u) = IC(u) \times CPI(u) \times Clock_cycle_time$
 $= IC(u) \times 1.57 \times 2 \times 10^{-9} = 3.14 \times 10^{-9} \times IC(u)$
- Optimized
 - $CPI(o) = [(0.43/2) \times 1 + 0.57 \times 2] / (1 - 0.43/2) = 1.73$
 - $MIPS(o) = 500MHz / (1.73 \times 10^6) = 289.0$
 - $CPU_time(o) = IC(o) \times CPI(o) \times Clock_cycle_time$
 $= 0.785 \times IC(u) \times 1.73 \times 2 \times 10^{-9} = 2.72 \times 10^{-9} \times IC(u)$

05/01/2004 UAH-CPE631 32

Things to Remember

- Execution, Latency, Res. time: time to run the task
- Throughput, bandwidth: tasks per day, hour, sec
- User Time
 - time user needs to wait for program to execute: depends heavily on how OS switches between tasks
- CPU Time
 - time spent executing a single program: depends solely on design of processor (datapath, pipelining effectiveness, caches, etc.)

Things to Remember (cont'd)

- Benchmarks: good products created when have good benchmarks

- CPI Law

$$CPU\ time = \frac{Instructions}{Program} \cdot \frac{Clockcycles}{Instruction} \cdot \frac{Seconds}{Clockcycle} = \frac{Seconds}{Program}$$

- Amdahl's Law

$$Speedup = \frac{1}{1 - Fraction_{enhanced} + \frac{Fraction_{enhanced}}{Speedup_{enhanced}}}$$

Appendix #1

Why not Arithmetic Mean of Normalized Execution Times

Program	Ref. Com.	Com. A	Com. B	Com. C	A/Rel	B/Rel	C/Rel
P1 (sec)	100	10	20	5	0.1	0.2	0.05
P2 (sec)	10 000	1000	500	2000	0.1	0.05	0.2
Total (sec)	10100	1010	520	2005			
AM (w1=w2=0.5)	5050	505	260	1002.5	0.1	0.125	0.125
GM					0.1	0.1	0.1

AM of normalized execution times; do not use it!

Problem: GM of normalized execution times rewards equally all 3 computers?