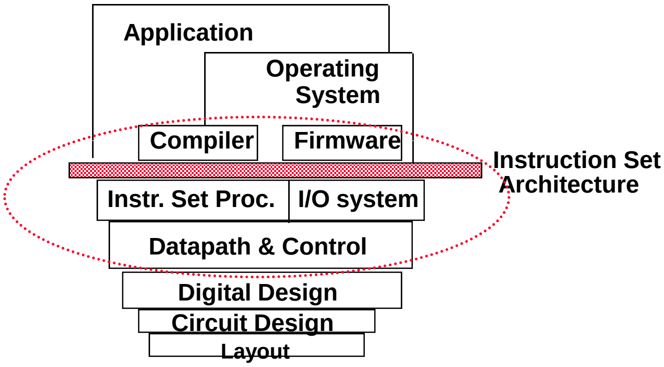
CS4200/5200
Computer Architecture I

Lecture 2: Quantitative Performance Evaluation

Dr. Xiaobo Zhou
Department of Computer Science

Review: What is "Computer Architecture"?

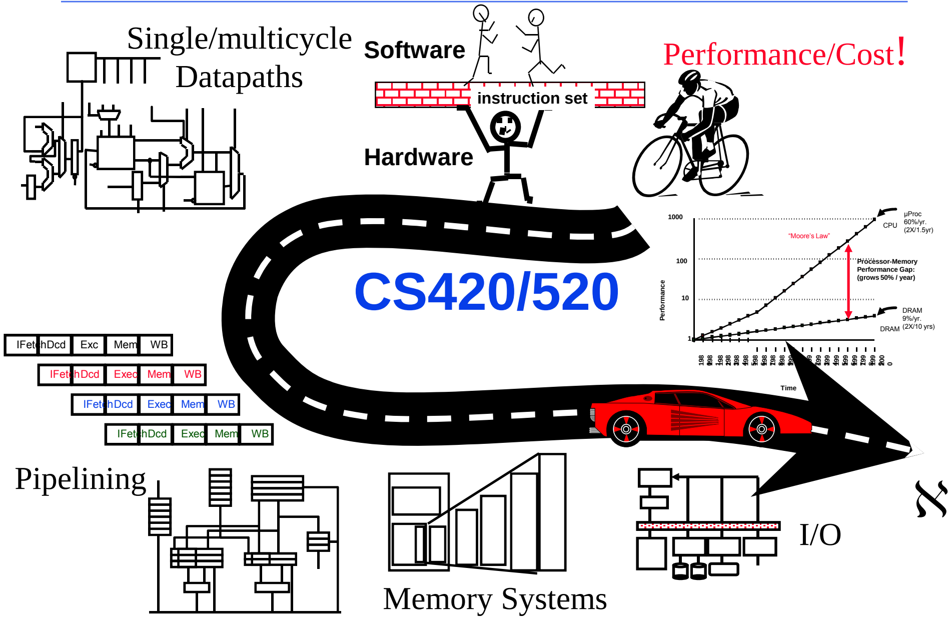


- Coordination of many *levels of abstraction*
- Under a rapidly changing set of forces
- Design, Measurement, *and* Evaluation

Re: Summary of Lecture 1

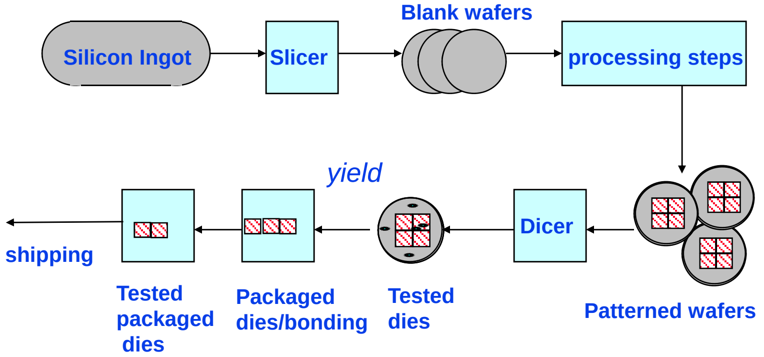
- Trends in Technology and Performance
- Computer Architecture: ISA + Organization + Hardware
- ISA: RISC vs. CISC
- All computers consist of five components
  - Processor: (1) datapath and (2) control
  - (3) Memory
  - (4) Input devices and (5) Output devices
- Not all “memory” are created equally
  - Cache: fast (expensive) memory are placed closer to the processor
  - Main memory: less expensive memory--we can have more
- Input and output (I/O) devices has the messiest organization
  - Wide range of speed: graphics vs. keyboard
  - Wide range of requirements: speed, standard, cost ... etc.
  - Least amount of research (so far)

Where Are We ??

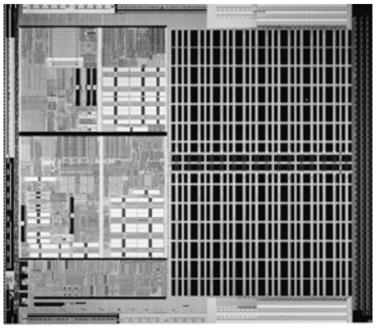


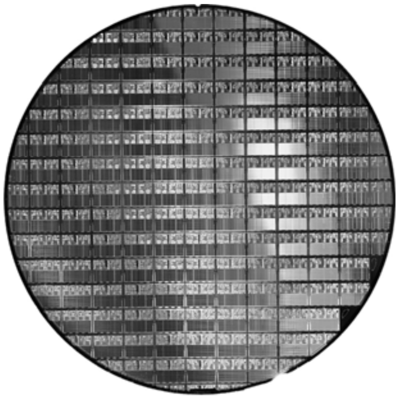
Cost: Chip Manufacturing Process

- Silicon (semiconductor) can be transformed with materials to
  - Conductors, insulators, on/off switch (transistor)
- VLSI (very large-scale *integrated circuit*)
  - Millions of combinations, manufactured in a single package
  - Critical to the cost of the chips and machines



Real World Examples



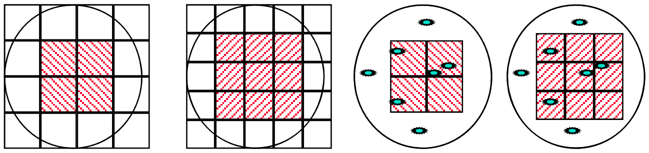


- Left: an AMD Opteron microprocessor die
- Right: an 300mm wafer contains 117 AMD Opteron chips

Die Costs

$$\text{Die cost} = \frac{\text{Wafer cost}}{\text{Dies per Wafer} * \text{Die yield}}$$

$$\text{Dies per wafer} = \frac{\pi * (\text{Wafer\_diam} / 2)^2}{\text{Die Area}} - \frac{\pi * \text{Wafer\_diam}}{\sqrt{2 * \text{Die Area}}} - \text{Test dies} \approx \frac{\text{Wafer Area}}{\text{Die Area}}$$



$$\text{Die Yield} = \frac{\text{Wafer yield (1)}}{(1 + \text{Defects\_per\_unit\_area} * \text{Die area})^{N}}$$

1) Defects per unit area is a measure of the random manufacturing defects. In 2010, the value was typically 0.016 to 0.057 defects per cm^2.
2) N is a parameter called the process-complexity factor, a measure of manufacturing difficulty. In 2010, N ranges from 11. to 15.5.

Example 1: Dies per Wafer

Find the maximum number of dies per 30cm-diameter wafer for a die that is 1.5 cm on a side.

Answer:

Dies per wafer ≈ wafer area / die Area

die area = 1.5 cm * 1.5 cm = 2.25 cm^2

wafer area = π * (30/2)^2 = 706.9 cm^2

Dies per wafer = 706.9 / 2.25 = 314

More accurately:

Dies per wafer = 706.9 / 2.25 – 94.2 / 1.41 = 270

Example 2: Die Yield

Find the die yield for dies that are 1.5 cm on a side and 1.0 cm on a side, respectively. Assuming a defect density of 0.031 per cm^2 and parameter ***N*** = 13.5. For simplicity, the wafer yield is assumed to be 100%.

$$\text{Die yield} = \frac{\text{Wafer yield}}{(1 + \text{Defects\_per\_unit\_area} \; * \text{Die area})^{N}}$$

Answer:

The die areas are 2.25 cm^2 and 1.0 cm^2, respectively.

For the larger die, the yield is (1+ 0.031 x 2.25)^13.5 = 0.4

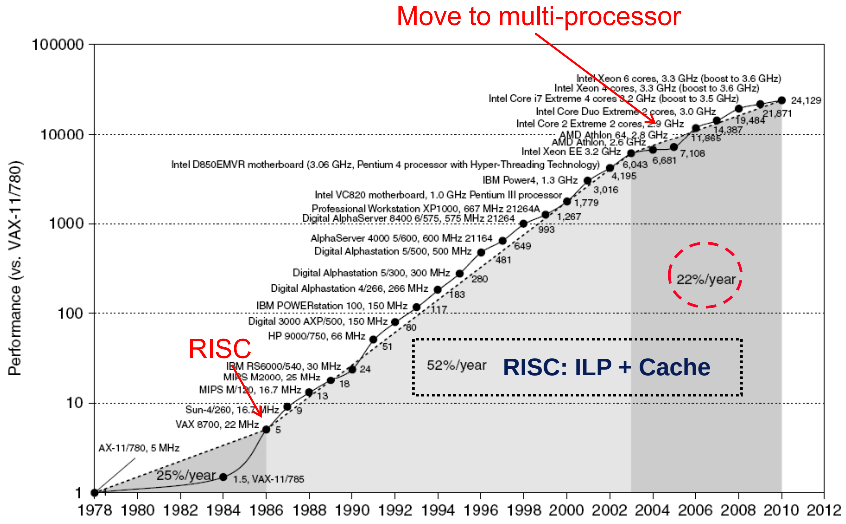
For the smaller die, the yield is (1+0.031 x 1.0)^13.5 = 0.66

Integrated Circuit Costs

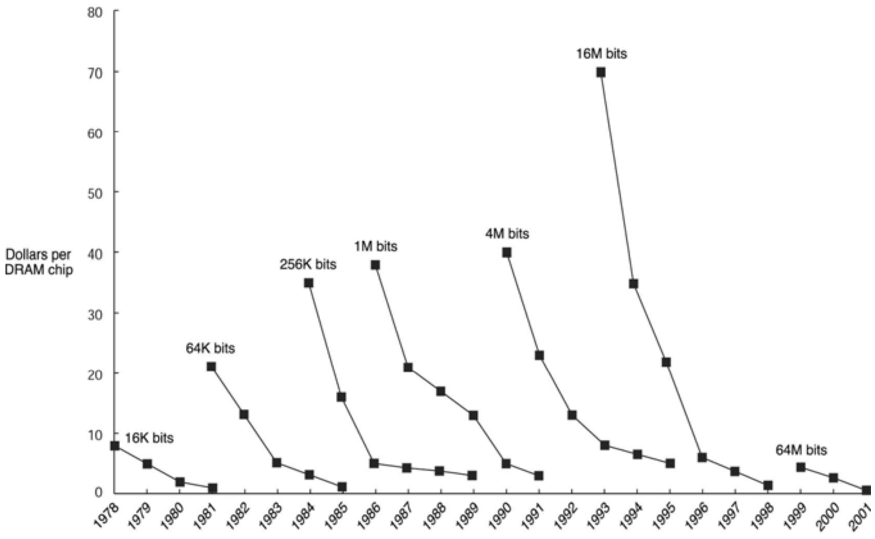
$$\text{IC cost} = \frac{\text{Die cost} + \text{Testing cost} + \text{Packaging cost}}{\text{Final test yield}}$$

Chip	***Die cost***	***Package cost***	***Test & Assembly***	***Total***
386DX	**$4**	**$1**	**$4**	**$9**
486DX2	**$12**	**$11**	**$12**	**$35**
PowerPC 601	**$53**	**$3**	**$21**	**$77**
HP PA 7100	**$73**	**$35**	**$16**	**$124**
DEC Alpha	**$149**	**$30**	**$23**	**$202**
SuperSPARC	**$272**	**$20**	**$34**	**$326**
Pentium	**$417**	**$19**	**$37**	**$473**

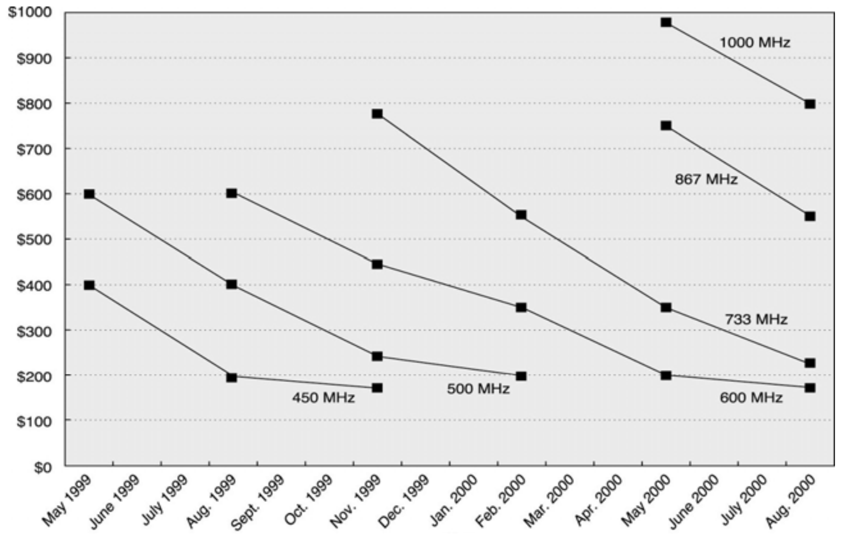
Re: Processor Performance (SPEC)



Price of Six Generations of DRAMs



Price of an Intel Pentium III



Performance: Two notions of “performance”

Plane	DC to Paris	Speed	Passengers	Throughput (pmph)
Boeing 747	6.5 hours	610 mph	470	286,700
BAD/Sud 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

Definitions of Performance

- Performance is in units of things-per-second
  - bigger is better
- If we are primarily concerned with response time
  - performance(x) = $\dfrac{1}{\text{execution\_time(x)}}$

" X is n times faster than Y" means

$$N = \frac{\text{Performance(X)}}{\text{Performance(Y)}} = \frac{\text{Execution\_time (Y)}}{\text{Execution\_time (X)}}$$

Time of Concorde vs. Boeing 747?
Concord is 1350 mph / 610 mph = 2.2 times faster
= 6.5 hours / 3 hours

We will focus primarily on execution time for a single job

Relating Processor Metrics

- CPU execution time = CPU clock cycles/pgm X clock cycle time
- or CPU execution time = CPU clock cycles/pgm ÷ clock rate
- CPU clock cycles/pgm = Instructions/pgm X avg. clock cycles per instr.
- or CPI = CPU clock cycles/pgm ÷ Instructions/pgm
- Different instructions may take different amounts of time (and/or different # of clock cycles) depending on what they do, CPI is an average of all the instructions executed in program
- CPI tells us something about the Instruction Set Architecture, the Implementation of that architecture (since the instruction count required for a program is the same)
- IPC = # instructions per clock cycle, the inverse of CPI

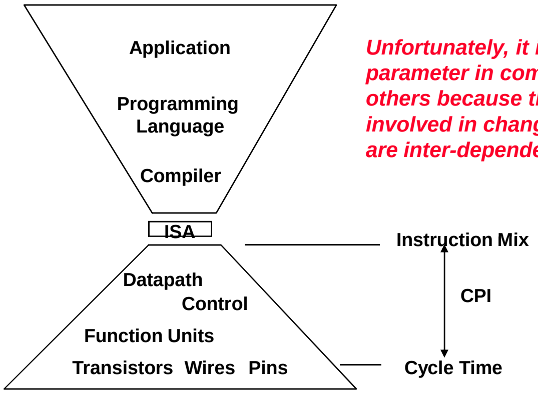
Aspects of CPU Performance

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

	instr. count	CPI	clock rate
Program	X	X	
Compiler	X	X	
Instr. Set.	X	X	
Organization		X	X
Technology			X

Organizational Trade-offs

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



Unfortunately, it is hard to change one parameter in complete isolation from others because the basic technologies involved in changing each component are inter-dependent!!!

Example: Clock Rate

Our favorite program runs in 10 sec on machine A, which has a 400MHz clock. We are trying to design a machine B with faster clock rate so as to reduce the execution time to 6 sec.

The increase of clock rate will affect the rest of the CPU design, causing B to require 1.2 times as many clock cycles as machine A for this program. What clock rate should be?

Answer:

CPU execution time = CPU clock cycles/pgm ÷ clock rate

CPU time A = CPU clock cycle A / clock rate A

==> CPU clock cycle A = 10 sec x 400 x 10^6

CPU time B = CPU clock cycle B / Clock rate B

Clock rate B = CPU clock cycle B / CPU time B
= 1.2*CPU clock cycle A / CPU time B
= 1.2*4000*10^6 / 6 = 800 MHz

CPI: Average Cycles per Instruction

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

CPI = (CPU Time * Clock Rate) / Instruction Count
= Clock Cycles of a program / Instruction Count

$$\text{CPU time} = \text{ClockCycleTime} * \sum_{i=1}^{n} \text{CPI}_i * I_i$$

"instruction frequency"

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

Example: CPI

Base Machine (Load/Store) and Instruction frequencies in the execution of a program:

Op	Freq	Cycles (per instruction)
ALU	**40%**	**1**
Load	**30%**	**2**
Store	**20%**	**2**
Branch	**10%**	**2**

Question: What is the average CPI of the program on the machine

Answer:

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

Example: Performance Comparison

Suppose we have two implementations of the same instruction set. Machine A has a clock cycle time of 10 ns and an average CPI of 2.0 for some program.

Machine B has a clock cycle time of 20 ns and an average CPI of 1.2 for the same program and compiler.

Which is faster? And by how much?

Let I denote the number of instructions of the program
CPU time A = I * 2.0* 10 = 20 I
CPU time B = I * 1.2 *20 = 24 I

Machine A is 1.2 times faster than B

Marketing Metrics

MIPS = Instruction Count / (ExTime * 10^6)

= Clock Rate / (CPI * 10^6)

- Million Instructions Per Seconds

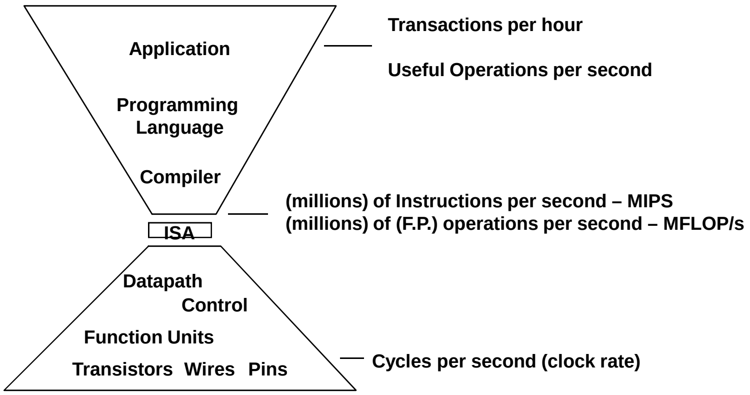
Three problems with using MIPS as a measure

- programs with different instruction mixes? no single MIPS!
- machines with different instruction sets? IC varies!
- uncorrelated with performance! Perhaps, inversely.

MFLOP/S = FP Operations / ExTime * 10^6

- Million Floating-point Operations Per Second
- machine and program dependent

Metrics of Performance


Application
Programming Language
Compiler
ISA
Datapath
Control
Function Units
Transistors Wires Pins
Transactions per hour
Useful Operations per second
(millions) of Instructions per second – MIPS
(millions) of (F.P.) operations per second – MFLOP/s
Cycles per second (clock rate)

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

Example: CPI & MIPS

Assume we build an optimizing compiler for the load/store machine. The compiler discards 50% of the ALU instructions.

1) What is the CPI_opt ?
2) Ignoring system issues and assuming a 20 ns clock cycle time (50 MHz clock rate). What is the MIPS rating for optimized code versus un-optimized code? Does the MIPS rating agree with the rating of execution time?

Answer: **MIPS = Instruction Count / Time * 10^6**

= Clock Rate / CPI * 10^6

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

Why Do Benchmarks?

- How we evaluate differences
  - Different systems
  - Changes to a single system
- Benchmarks are programs specially chosen to measure performance.
  - Benchmarks should represent large class of important programs (say engineering environments)
  - Improving benchmark performance should help many programs
- For better or worse, benchmarks shape a field
  - Good ones accelerate progress
  - Bad benchmarks hurt progress
- The best type of programs to use for benchmarks are real applications.

Programs to Evaluate Processor Performance

- Synthetic Benchmarks
  - artificial programs, attempt to match the characteristics of a large set of real programs
  - e.g., Whetstone, dhrystone
- (Toy) Benchmarks
  - execute in a small code segment, usually, 10-100 line
  - e.g.,: sieve, puzzle, quicksort
- Kernel benchmarks
  - small, time-intensive pieces extracted from real programs
  - primarily for benchmarking high-end machines, supercomputers
  - e.g., Livermore loops, linpack
- Modified applications
- Real applications
  - e.g., gcc, spice

Increasing order of accuracy of perf. prediction ↓

Successful Benchmarks: SPEC

- 1987 RISC industry mired in “bench marketing”:
  (“That is 8 MIPS machine, but they claim 10 MIPS!”)

- 5 companies band together to perform Systems Performance Evaluation Committee (SPEC) in 1988:
  Sun, MIPS, HP, Apollo, DEC

- SPEC was created to improve the measurement and reporting of CPU performance, through a better controlled measurement process and the use of more realistic benchmarks.

- SPEC created standard list of programs, inputs, reporting: some real programs, includes OS calls, some I/O-intensive activities.

- CPU-intensive benchmarks and graphics-intensive benchmarks
  - SPEC CPU2000 and SPECapc^{SM}

- More details: http://www.spec.org

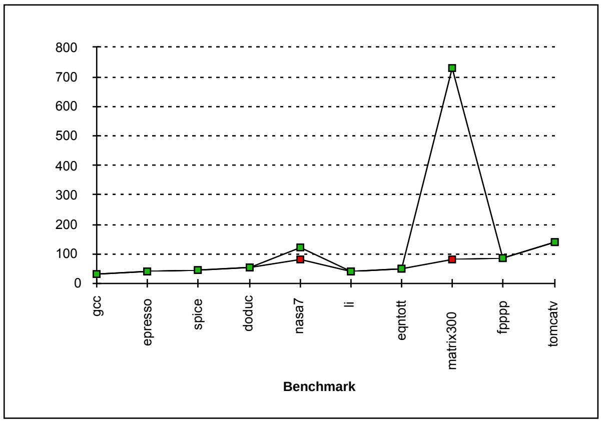
Other Benchmarks: TPC and EEMBC

- TPC: Transaction Processing Council (www.tpc.org)
  - Transaction-processing (TP) benchmarks measure the ability of a system to handle transactions, i.e., DB accesses and updates
    - airline reservation systems or banking ATM systems
  - TPC-A (1985): the first benchmark
  - TPC-C (1992): a complex query environment
  - TPC-H: ad hoc decision systems – queries are unrelated
  - TPC-R: a business decision support system, standard queries
  - TPC-W: a Web-based transaction benchmark

- EEMBC: the EDN Embedded Microprocessor Benchmark Consortium
  - Embedded benchmarks for embedded computing systems
  - “embassy”

SPEC First Round

- First round 1989; 10 programs, single number to summarize performance (inverse to execution time)
- One program: 99% of time in single line of code
- New front-end compiler could improve dramatically



Fallacy: Benchmarks remain valid indefinitely

SPEC Evolution

- Second round; SpecInt92 (6 integer programs) and SpecFP92 (14 floating point programs)
  - Matrix300 was dropped
- Third round; 1995; new set of programs: 8 integer programs and 10 floating point programs
  - “Benchmarks useful for 3 years”
- Fourth round; SPEC CPU2000: CINT2000 (11 integer programs) and CFP2000 (14 floating-point benchmarks)
  - SPECweb99 for Web servers
  - Two graphics-intensive benchmarks:
    - SPECviewperf
    - SPECapc
- Fifth round; SPEC CPU2004 (http://www.specbench.org/cpu2004/)

How to Summarize Results?

	Computer A	Computer B
Program P1 (sec):	1	10
Program P2 (sec):	1000	100

What are your conclusions?

" X is n times faster than Y" means

$$\frac{\text{ExTime(Y)}}{\text{ExTime(X)}} = \frac{\text{Performance(X)}}{\text{Performance(Y)}} = n$$

- **Machine A is 10 times faster than B for program P1**
- **Machine B is 10 times faster than A for program P2**

Total Execution Time: A Consistent Summary

	Computer A	Computer B
Program P1 (sec):	1	10
Program P2 (sec):	1000	100
Total time (sec):	1001	110
Arithmetic mean:	500.5	55

- **Total execution time**
  - **B is 1001/110=9.1 times faster than A**
- **Arithmetic mean: an average of the total execution time**

$$\frac{1}{n}\sum_{i=1}^{n} Time_i$$

Weighted Execution Time

	Computer A	Computer B
Program P1 (sec):	1	10
Program P2 (sec):	1000	100
Arithmetic mean:	500.5	55

Q: Are P1 and P2 run equally in the workload?

- **Weighted Arithmetic Mean**

$$\frac{1}{n}\sum_{i=1}^{n} Weight_i * Time_i$$

SPECRatio and Geometric Mean

	Computer A	Computer B
Program P1 (sec):	1	10
Program P2 (sec):	1000	100
Program P3 (sec):	100	25

- **SPECRatio: normalize execution time to the reference computer, but does not predict execution time**
  - **A ratio rather than an absolute execution time**
- **Geometric Mean**

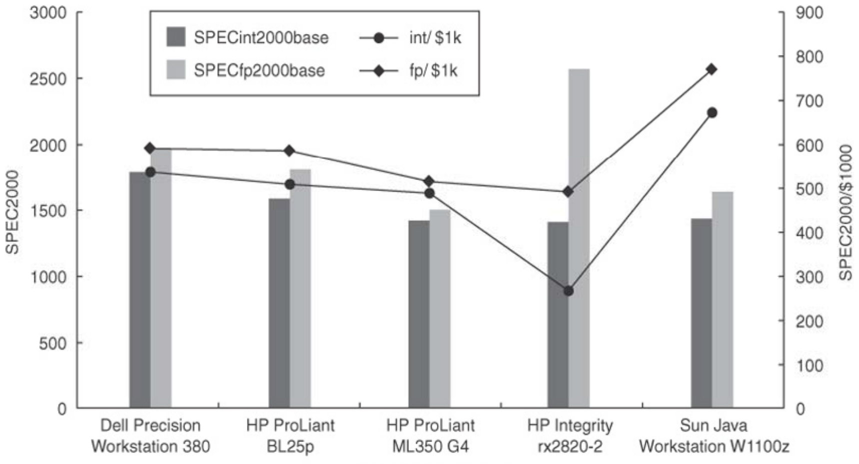
$$\sqrt[n]{\prod_{i=1}^{n} Execution\ time\ ratio_i} \qquad \left(\prod_{i=1}^{n} \alpha_i = \alpha_1 \times \alpha_2 \times \ldots \times \alpha_n\right)$$

Performance and Price-Performance

Vendor / model	Processor	Clock rate (GHz)	Price
Dell Precision Workstation 420	Intel P4 Xeon	3.8	$3,346
HP ProLiant BL25p	AMD Opteron 252	2.6	$ 3,099
HP ProLiant ML350 G4	Intel P4 Xeon	3.4	$ 2,907
HP Integrity rx2620-2	Itanium 2	1.6	$5,201
Sun Java WS W1100z	AMD Opteron 150	2.4	$2,145

° **Prices (as of Aug 2005): many factors are responsible to prices, including expandability, disk, memory, CPU, etc.**

Performance and Performance/Cost (Cont.)



© 2007 Elsevier, Inc. All rights reserved.

• Performance (as of Jan 2006): SPEC CINT2000 summarizes CPU performance; larger number indicating higher performance

- Does clock rate reflect the performance ?

Amdahl's Law -- Example:

Suppose a person wants to travel from city A to city B by city C. The routes from A to C are in mountains and the routes from C to B are in a desert. The distances from A to C, and from C to B are 80 miles and 200 miles, respectively.

From A to C, walk at speed of 4 mph
From C to B, walk or drive (at speed of 100 mph)

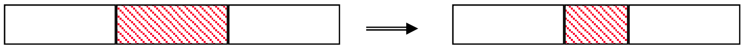
Question: How long will it take for the entire trip?
How much faster from A to B by a car as opposed to walk?

Quantitative Principles: Amdahl's Law

ExTime after improvement = ExTime unaffected + Extime affected / amount of improvement

Speedup due to enhancement E:

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



Amdahl's Law (Cont.)

Two key factors:

1) *The fraction of the original execution time can be improved*
   e.g., if 20s of the execution time of a program that takes 60s in total can use an enhancement, the fraction (F) = 20/60

2) *The improvement gained by the enhanced execution mode*
   e.g., if the enhancement mode takes 2s for some portion of the program that can completely use the mode, while the original mode takes 5s for the same portion, the improvement is 5/2.

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

ExTime(with E) = ((1-F) + F/S) X ExTime(without E)

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

Example: One Enhancement Factor

Suppose an enhancement make a processor runs 5 times faster than the original one, but is only usable 60% of the time

Question 1: what is the overall speedup?

Answer:

ExTime(with E) = ((1-F) + F/S) X ExTime(without E)

Speedup(with E) = ExTime(without E) ÷ ((1-F) + F/S) X ExTime(without E)

Fraction_enhance = 0.6
Speedup_enhanced = 5
Speedup_overall = 1/(0.4+0.6/5) ≈ 1.92

Q2: to have a speedup 2, how much faster the enhancement must run?
Q3: what is the maximum possible speedup?

Example: Multiple Enhancement Factors

You have a program that takes 100 seconds to execute. Of this time, 20 seconds for addition, 40 seconds for multiplication, 40 seconds For memory access instructions.

Enhancement A: make multiplication 4 times faster.
Enhancement B: make addition 2 times faster .

Question 1: what is the speedup if only A is used?
2: what is the speedup if both A and B are used?

Example: Comparing the speedups

A common transformation required in graphics processors is square root.

FPSQR is responsible for 20% of the execution time
FP operations (including FPSQR) is responsible for 50%

Alternative 1: speed up FPSQR by a factor of 10
Alternative 2: speed up all FP by a factor of 1.6

Q: which alternative is more effective for performance improvement?

Answer:

Speedup(with E) = ExTime(without E) ÷
((1-F) + F/S) X ExTime(without E)

Example: CPI measurements

Suppose we have made the following measurements:

Frequency of FP operations: 25%
Average CPI of FP operations: 4.0
Average CPI of the other instructions: 1.33

1) What is the CPI of the machine?
2) If we have also made the following measurements:

Frequency of FPSQR operations: 2%
Average CPI of FPSQR operations: 20

Now, we can decrease the CPI of FPSQR to 2 by a new design. What is the new CPI? And what is the speedup of the design?

Answer:

CPI = CPI_ori – 2% x (CPI _old FPSQR – CPI_new FPSQR)
= 2.0 – 2% x (20 – 2) = 1.64

Example: Compiling

Assume we build an optimizing compiler for the load/store machine.

load/store machine			After optimization	
Op	**Freq**	**CPI**	**Percentage of Instr. executed**	
ALU	**40%**	**1**	**50%**	**(50% discarded)**
Load	**30%**	**2**	**80%**	**(20% ")**
Store	**20%**	**2**	**90%**	**(10% ")**
Branch	**10%**	**2**	**100%**	**(0% ")**

1) What is the new CPI?
2) What is the speedup by the use of the new compiler?

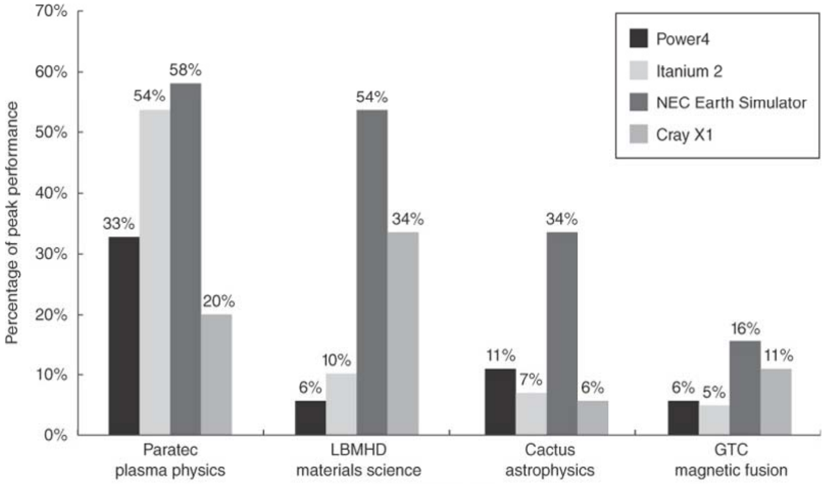
Fallacies

° *The cost of the processor dominates the cost of the system*

Vendor / model	Processor + cabinetry	Memory	Storage	Software
IBM eServer p5 595/64	28%	16%	51%	6%
IBM eServer p5 595/32	13%	31%	52%	4%
HP Integrity rx5670 cluster	11%	22%	35%	33%
HP Integrity Superdome	33%	32%	15%	20%
IBM eServer pSeries 690	21%	24%	48%	7%
Median of HPC	21%	24%	48%	7%
Dell PowerEdge 2800	6%	3%	80%	11%
Dell PowerEdge 2850	7%	3%	76%	14%
HP ProLiang ML350/1	5%	4%	70%	21%
HP ProLiang ML350/2	9%	8%	65%	19%
HP ProLiang ML350/3	8%	6%	65%	21%
Median of desktops	7%	4%	70%	19%

Fallacies

° *Peak performance tracks observed performance (car mileage)*



© 2007 Elsevier, Inc. All rights reserved.

Performance Evaluation Summary

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

° Integrated circuits driving computer industry

° Die costs goes up with the cube/quad of die area

° Time is the only valid measure of computer performance!

° Good products created when have:

- Good benchmarks
- Good ways to summarize performance

° Remember Amdahl’s Law

- Speedup is limited by unimproved part of program

° More reading

- CA4: Chapter 1 (reference CO3’s Chapter)

Reading and Homework

• Reading:

CA 5: Chapter 1 (or CA 4 - Chapter 1)

CO 4: Chapter 1

• Homework, due 1 week later from the release date, see course Web site

•Preview:

CO 4: Chapter 2 (MIPS)

CA 5: Appendix A (ISA)