EEL-4713
Computer Architecture
Performance

Overview of Today's Lecture: Performance

° Definition and Measures of Performance

° Summarizing Performance and Performance Pitfalls

° Reading: Chapter 1

Technology and Cost Summary

° Integrated circuits driving computer industry

° Technology improvements:
- CMOS transistors getting smaller, faster for each new generation
- Smaller -> more transistors per area -> more functionality (e.g. 64-bit datapath, MMX extensions, superscalar execution, caches, multiple cores)
- Faster -> higher raw speed (clock cycle)

° Die costs goes up with the cube of die area

Review: Summary from Chapter 1

° 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, limited amount due to large area
- Main memory: less expensive memory--we can have more

° Input and output (I/O) devices are very diverse
- Wide range of speed: graphics vs. keyboard
- Wide range of requirements: speed, standard, cost ... etc.

Performance

° Purchasing perspective
  - given a collection of machines, which has the
    - best performance ?
    - least cost ?
    - best performance / cost ?

° Design perspective
  - faced with design options, which has the
    - best performance improvement ?
    - least cost ?
    - best performance / cost ?

° Both require
  - basis for comparison
  - metric for evaluation

° Our goal is to understand cost & performance implications of architectural choices

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

Example

- Time of Concorde vs. Boeing 747?
  - Concord is 1350 mph / 610 mph = 2.2 times faster
    = 6.5 hours / 3 hours
- Throughput of Concorde vs. Boeing 747 ?
  - Concord is 178,200 pmph / 286,700 pmph = 0.62 "times faster"
  - Boeing is 286,700 pmph / 178,200 pmph = 1.6 "times faster"

- Boeing is 1.6 times ("60%") faster in terms of throughput
- Concord is 2.2 times ("120%") faster in terms of flying time

We will focus primarily on execution time for a single job

Definitions

° Performance is in units of things-per-second
  - bigger is better

° If we are primarily concerned with response time
  - performance(x) = $\frac{1}{\text{execution\_time(x)}}$

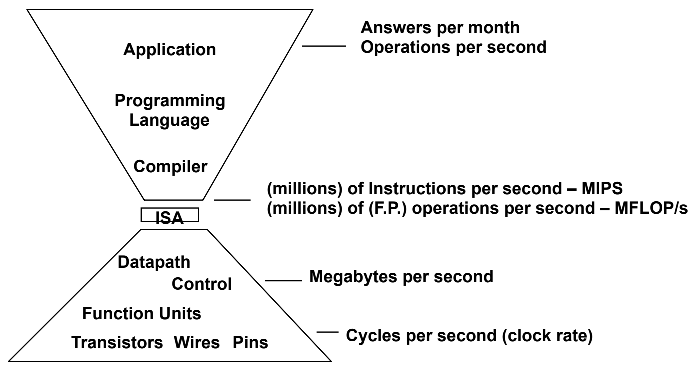
" X is n times faster than Y" means

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

Basis of Evaluation

Pros		Cons
• representative	Actual Target Workload	• very specific • non-portable • difficult to run, or measure • hard to identify cause
• portable • widely used • improvements useful in reality	Full Application Benchmarks	•less representative than target workload
• easy to run, early in design cycle	Small "Kernel" Benchmarks	• easy to "fool"
• identify peak capability and potential bottlenecks	Microbenchmarks	• "peak" may be a long way from application performance

Metrics of performance



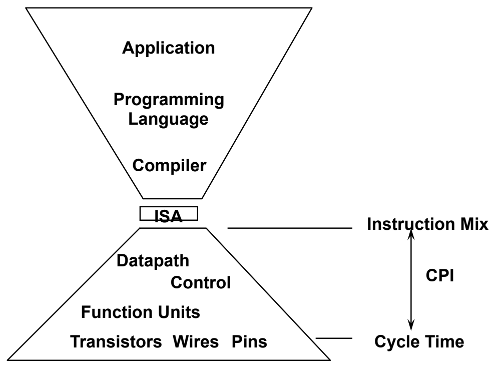
Relating Processor Metrics

° **CPU execution time = CPU clock cycles/program * clock cycle time**
  - **Or, CPU execution time = CPU clock cycles/program ÷ clock rate**

° **CPU clock cycles/program = Instructions/program * avg. clock cycles per instruction**
  - **Or, more commonly: CPI (clock cycles per instruction) = (CPU clock cycles/program) ÷ (Instructions/program)**

° **Examples:**
  - **Single-cycle MIPS datapath: CPI=1 (all instructions take 1 cycle)**
  - **Multi-cycle MIPS datapath: CPI within 2-5 range**

Organizational Trade-offs



CPI

CPI varies depending on individual instruction

e.g. 5-cycle load, 3-cycle ALU in multi-cycle MIPS

Average CPI for a program depends on the "*mix*" of instructions it executes

e.g. close to 5 if load-intensive, close to 4 if ALU intensive

n classes/types of instructions

The instruction mix in %

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

Example (RISC processor)

Base Machine

Op	**Freq**	**Cycles**	**F*CPI(i)**	**% Time**
ALU	**50%**	**1**	**.5**	**23%**
Load	**20%**	**5**	**1.0**	**45%**
Store	**10%**	**3**	**.3**	**14%**
Branch	**20%**	**2**	**.4**	**18%**
			2.2	

Typical Mix

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 shave a cycle off the branch time?

What if two ALU instructions could be executed at once?

Invest resources where time is spent!

Aspects of CPU Performance

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

	IC instr. count	**CPI** CPI	**CLK** clock rate
Program			
Compiler			
Instr. Set. Arch.			
Organization			
Technology			

Aspects of CPU Performance

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

	IC instr. count	**CPI** CPI	**CLK** clock rate
Program	**X**	**(x) avg**	
Compiler	**X**	**(x) avg**	
Instr. Set. Arch.	**X**	**X**	**X**
Organization		**X**	**X**
Technology			**X**

Marketing Metrics

MIPS = Instruction Count / Time * 10^6

= Clock Rate / CPI * 10^6

•machines with different instruction sets ?

•programs with different instruction mixes ?

  • dynamic frequency of instructions

• uncorrelated with performance

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

•machine dependent

•Not necessarily where time is spent

Why benchmarks?

° How we evaluate differences
  • Different systems
  • Changes to a single system

° Provide a target
  • Benchmarks should represent large class of important programs
  • Improving benchmark performance should help many programs

° For better or worse, benchmarks shape a field

° Good ones accelerate progress
  • good target for development

° Bad benchmarks hurt progress
  • New ideas that help real programs v. sell machines/ papers?

Programs to Evaluate Processor Performance

° (Toy) Benchmarks
  • 10-100 line
  • e.g.,: sieve, puzzle, quicksort, “cast”

° Synthetic Benchmarks
  • attempt to match average frequencies of real workloads
  • e.g., Whetstone, dhrystone

° Kernels
  • Time critical excerpts

Successful Benchmark: SPEC

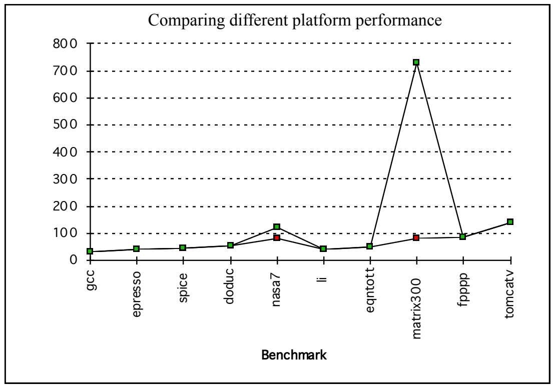
° 1987: RISC industry mired in “bench marketing”
  • Inconistent
  • Not reported fairly or correctly
  • Everyone had a “new and better” benchmark targeted to make their architecture look better

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

° Create standard list of programs, inputs, reporting: some real programs, includes OS calls, some I/O

SPEC first round

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



SPEC Evolution

- Second round; SpecInt92 (6 integer programs) and SpecFP92 (14 floating point programs)
- Third round; 1995; new set of programs
- Currently: SPEC 2006
- Additions:
  - SPECweb (Web server throughput)
  - JVM (Java virtual machine)
  - SPEChpc (high-performance computing)
  - SFS (file system)
  - SPECviewperf, SPECapc (graphics)

http://www.spec.org

Quantitative design principles

- Primary goal: cost-performance
  - Increase performance with small cost implications
- Key principle to keep in mind:
  - *"Make the common case fast"*
  - Quantified by "Amdahl's Law"

Amdahl's Law

- Idea:
  - Given a system "X" and the opportunity of enhancing it to become a new system "Y"
  - How faster will "Y" be relative to "X"?
- Key parameters:
  - *The gain* from the enhancement
  - *The frequency* at which it can be applied

Example - Triathlon

- Three parts: run, swim, cycle
- Bob has been training for a competition
  - Based on his experience, he knows that during competition he can push his limit to:
    - Run 30% faster than in training, *or*
    - Swim 50% faster, *or*
    - Cycle 20% faster
- Where should Bob spend his energy during competition?
- By the way, when training Bob spends:
  - 60 minutes running
  - 40 minutes swimming
  - 2 hours cycling

Amdahl's Law

- Used to compute *speedups:*

```
                  Performance_with_enhancement
Speedup = ----------------------------------------------
                  Performance_without_enhancement
```

- Performance:
  - Inversely proportional to execution time
  - Speedup = EXECold/EXECnew

Amdahl's Law (cont)

EXECnew = EXECold *

[(1 – FRACenh) + FRACenh/SPenh]

EXECnew,old:

execution times (seconds)

FRACenh:

Fraction of time enhancement is applied (%)

SPenh:

Speedup due to enhancement (absolute number)

Example

- Bob's speedup due to swimming:
  - FRACenh = 40 min/220 min = 0.182
  - SPenh = 1.50 (50% speedup)
  - Speedup = EXECold/EXECnew =
    = 1/[(1-0.182)+0.182/1.50]
    = 1.065 (6.5% improvement)
- Running: 1.067 (6.7% improvement)
- Cycling: 1.100 (10% improvement)

Other important design principles

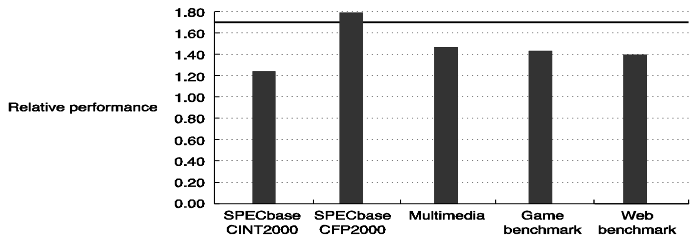
° Locality
  - Programs tend to reuse code/data recently accessed
    - Memory hierarchies leverage this locality for increased performance
    - Combats the memory wall

° Parallelism
  - Multiple operations in a single clock cycle
    - Pipelining, super-scalar execution, multi-core designs, vector processors

Fallacies & Pitfalls

° *Relative perf. can be judged by clock rates*
  - Fail to capture IC, CPI components
  - Cannot use clock rate to judge, even if same program, same ISA
    - IC same, but CPI may not be

e.g: Pentium 4 1.7GHz relative to P-III 1GHz



© 2003 Elsevier Science (USA). All rights reserved.

Summary

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

° Time is the measure of computer performance!

° Remember Amdahl's Law: Speedup is limited by unimproved part of program