Computer Architecture: Measuring Performance

Berk Sunar Thomas Eisenbarth

WPI

Last Time:
What is Computer Architecture?

A hardware perspective:

- Understand modern computer components and their interaction
- Learn to evaluate and compare computers
- Enable to make *own* design decisions

→Build your own computer

  - Instruction Set Design
  - Functional Organization
  - Logic Design
  - Implementation

Performance Measurement

Much of the focus of this course is on improving performance

Topics:

- performance metrics
- CPU performance equation
- benchmarks and benchmarking
- reporting averages
- Amdahl's Law
- Little's Law
- concepts
  - balance
  - tradeoffs
  - bursty behavior (average and peak performance)

Measuring Performance

Time to execute a program is the ultimate metric to determine performance

Execution time: t

(measured as: CPU time *or* wall time/response time (including I/O))

Performance: $\text{Perf} = \frac{1}{t}$

Comparing Performance:

A is N times faster than B iff: $\frac{\text{perf}_A}{\text{perf}_B} = \frac{t_B}{t_A} = N$

A is X% faster than B iff: $\frac{\text{perf}_A}{\text{perf}_B} = \frac{t_B}{t_A} = 1 + X/100$

Metrics of Performance

- "Time to execute a program" is the ultimate metric in determining the performance
- However, it is convenient to inspect other metrics as well when we examine the details of a machine.
- Computers use a clock that runs at a constant rate and determines when an event takes place in hardware.
- These discrete time intervals are called clock cycles (or ticks, clock ticks, clock periods).
- Clock rate (frequency) is the inverse of clock period.

Performance Metric I: MIPS

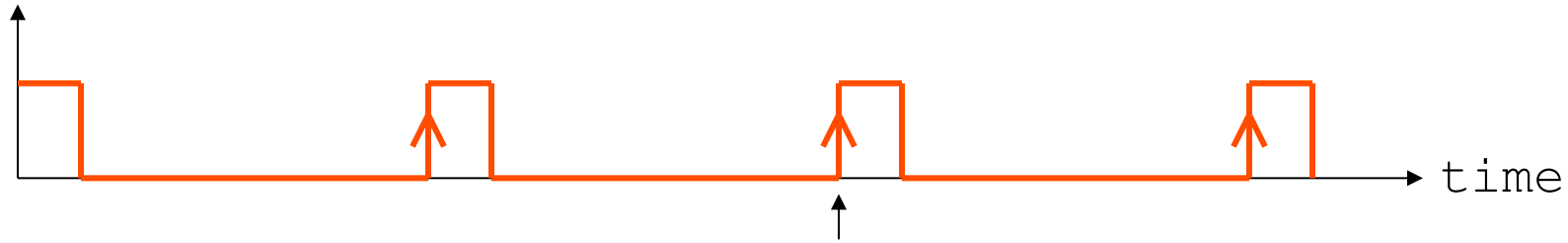
MIPS (millions of instructions per second):

$$\frac{\text{instruction count}}{\text{execution time in seconds}} \times 10^{-6}$$

- instruction count is not a reliable indicator of work
- may vary inversely with actual performance
- particularly bad metric for multicore chips

Clock Cycles

- Clock "ticks" indicate when to start activities



Start of events often the rising edge of the clock

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

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

The CPI Metric

- CPI Cycles Per Instruction
  - Number of cycles spent on an instruction on average.
  - Cycle count: $CC = IC \times CPI$
  - Hard to compute.
  - It is useful when comparing the performances of two machines with the same ISA. (Why?)

- Example: two machines with the same ISA. For a certain program we have
  - Machine A: CPI = 2.0
  - Machine B: CPI = 1.2
  - Which machine is faster?
  - What if machine A uses 250 ps and machine B 500 ps cycle time

CPU Performance Equation

Execution time = seconds/program (= 1 / Performance)

$$seconds/program = instructions/program \times cycles/instructions \times seconds/cycle$$

Instructions/program: dynamic instruction count; determined by ISA, compiler, and program

Cycles/instruction (CPI): CPI determined by ISA and micro-architecture (i.e. processor designer)

Seconds/cycle: clock time; determined by technology and CPU organization

CPU Performance Equation

Execution time = seconds/program (= 1 / Performance)

$$seconds/program = instructions/program \times cycles/instructions \times seconds/cycle$$

So, to improve performance

1. Increase the clock frequency (i.e. decrease the clock period)
2. Reduce the number of the clock cycles per instruction(CPI)
3. Reduce the number of instructions per program (IC)

CPU Performance Comparison

Example: "RISC Wars" (RISC vs. CISC)

assume

- instructions / program (IC): CISC = P, RISC = 2P
- CPI: CISC = 8, RISC = 2
- T = clock period for CISC and RISC (assumed equal)
- Recall: Exec. *Time* $= \mathrm{IC} \times \mathrm{CPI} \times \mathrm{T}$

➢ CISC time = P x 8 x T = 8PT

➢ RISC time = 2P x 2 x T = 4PT

➢ RISC time = CISC CPU time/2

CPU Back-of-the-Envelope Calculation

Example:

base machine

- 43% ALU ops (1 cycle), 21% loads (1 cycle), 12% stores (2 cycles), 24% branches (2 cycles)

  Note: pretending latency is 1 because of pipelining

Q: should 1-cycle stores be implemented if it slows clock 15%?

- Old CPI = 0.43 + 0.21 + **(0.12 x 2)** + (0.24 x 2) = 1.36
- New CPI = 0.43 + 0.21 + **0.12** + (0.24 x 2) = 1.24
- Speedup = (IC x 1.36 x T) / (IC x 1.24 x 1.15T) = 0.95

→Answer: NO!

How to Measure Performance?

How are execution-time & CPI *actually* measured?

- execution time: time (Unix cmd): wall-clock, CPU, system
- CPI=$CPU\ time/clock\ freq. \times instruction\ count$
- more useful? CPI breakdown (compute, memory stall, etc.)
  - so we know what the performance problems are (what to fix)

measuring CPI breakdown

- hardware event counters (built into core)
  - calculate CPI using instruction frequencies/event costs
- cycle-level microarchitecture simulator (e.g., SimpleScalar)
  + measure exactly what you want
  – model microarchitecture faithfully (at least parts of interest)
  →method of choice for many architects

Benchmarking

Program as unit of work:

- millions of them, many different kinds
- Which ones to use?

Benchmarks:

- standard programs for measuring/comparing performance
+ represent programs people care about
+ repeatable!!

Benchmarking process:

- define workload
- extract benchmarks from workload
- execute benchmarks on candidate machines
- project performance on new machine
- run workload on new machine

Benchmark: Which Programs to Choose?

Real programs:

+ only accurate way to characterize performance
- requires considerable work (porting)

Standard Performance Evaluation Corporation (SPEC)

- http://www.spec.org
- collects, standardizes and distributes benchmark suites
- consortium made up of industry leaders
- SPEC CPU (CPU intensive benchmarks):
  - SPEC89, SPEC92, SPEC95, SPEC2000, SPEC2006
- other benchmark suites:

  SPECjvm, SPECmail, SPECweb

Content of SPEC CPU 2006

12 integer programs (C, C++)

- gcc (compiler), perl (interpreter), hmmer (markov chain)
- bzip2 (compress), go (AI), sjeng (AI)
- libquantum (physics), h264ref (video)
- omnetpp (simulation), astar (path finding algs)
- xalanc (XML processing), mcf (network optimization)

17 floating point programs (C, C++, Fortran)

- *fluid dynamics*: bwaves, leslie3d, ibm
- *quantum chemistry*: gamess, tonto
- *physics*: milc, zeusmp, cactusADM
- gromacs (biochem)
- namd (bio, molec dynamics), dealII (finite element analysis)
- soplex (linear programming), povray (ray tracing)
- calculix (mechanics), GemsFDTD (computational E&M)
- wrf (weather), sphinx3 (speech recognition)

On Average Performance

Averages: sometimes confusing

- important things about averages (i.e., means)

Should be proportional to execution time (ultimate metric):

- Arithmetic Mean (AM) for times
- Harmonic Mean (HM) for rates (IPCs)
- Geometric Mean (GM) for ratios (speedups)

What does the Mean mean?

- Arithmetic mean (AM): average execution times of N programs:

$$\frac{1}{N}\sum_{1\ldots N} time_i$$

- Harmonic mean (HM): average IPCs of N programs:

$$\frac{N}{\sum_{1\ldots N}\frac{1}{\mathrm{IPC}_i}}$$

Example: 30 MPH for 1 mile + 90 MPH for 1 mile ≠ avg. 60 MPH

- Geometric mean (GM): average speedups of N programs:

$$\left(\prod_{1\ldots N} speedup_i\right)^{\frac{1}{N}}$$

Geometric Mean: Averaging Ratios

Example:

Execution time	Machine A	Machine B
Program 1	1	10
Program 2	1000	100

Compute Speedup:

Speedup *N*	Machine A	Machine B
Program 1	10	.1
Program 2	.1	10

Means:

- A is 10x faster than B for program 1
- B is 10x faster than A for program 2

Geometric Mean: Averaging Ratios

Compute Arithmetic mean of Speedup:

$10+.1/2 = 5.05$

- A is 5.05x faster than B on average
- B is 5.05x faster than A on average

Great fun, isn't it?

Compute Harmonic mean of Speedup:

$$\frac{2}{\frac{1}{10}+\frac{1}{.1}} = 5.05^{-1} = .198$$

- B is 5.05x faster than A on average
- A is 5.05x faster than B on average

Geometric Mean: Averaging Ratios

Compute Geometric mean of Speedup:

$$\sqrt[2]{.1 \times 10} = 1$$

- Geometric mean does **not** depend on base machine
- However, geometric mean of ratios is not proportional to total time!
  - For total execution time, B is 9.1 times faster

Amdahl's Law

Validity of the Single-Processor Approach to Achieving Large-Scale Computing Capabilities [G. Amdahl, AFIPS, 1967]

Let optimization speed up fraction *f* of program by factor *s*, then:

$$\text{Speedup} = \frac{t_{old}}{(1-f)\cdot t_{old} + \frac{f}{s}\cdot t_{old}} = \frac{1}{1-f+f/s}$$

Examples:

- f = 95%, s = 1.1 → 1/[(1-0.95) + (0.95/1.1)] = 1.094
- f = 5%, s = 10 → 1/[(1-0.05) + (0.05/10)] = 1.047
- f = 5%, s = ∞ → 1/[(1-0.05) + (0.05/ ∞)] = 1.052
- f = 95%, s = ∞ → 1/[(1-0.95) + (0.95/ ∞)] = 20

→ **make common case fast**, but...

...uncommon case eventually limits performance

Little's Law

Key Relationship between latency and bandwidth:

Average number in system = arrival rate × mean holding time

- Useful in design of computers, software, industrial processes, etc.

Example:

- How big should parking lot of grocery store be?
- Two customers per minute enter the store
- On average, each customer spends 30 minutes in store
- Lot size= 2 customers/minute * 30 minutes (* 1 car/customer)
- = 60 lots

System Balance

each system component produces & consumes data

- make sure data supply and demand is balanced
- X demand >= X supply ⇒ computation is "*X-bound*"
  - e.g., memory bound, CPU-bound, I/O-bound
- goal: be bound everywhere at once (why?)
- X can be bandwidth or latency
  - X is bandwidth ⇒ buy more bandwidth
  - X is latency ⇒ much tougher problem

Bursty Behavior

Question: to sustain 2 IPC, how many instructions should processor be able to

- fetch per cycle?
- execute per cycle?
- complete per cycle?

Answer: more than 2

- dependences will cause stalls (under-utilization)
- if desired performance is X, peak performance must be > X

programs don't always obey "average" behavior

→can't design processor only to handle average behavior

Next time:

Review of:

Instruction Set Architecture (ISA)