ECE437: Introduction to Digital Computer Design

Chapter 1b (performance, 1.4 onwards)

Fall 2016

Performance of Computers

- Which computer is fastest?
- Not so simple
  - scientific simulation - FP performance
  - program development - Integer performance
  - commercial work – memory + I/O

Performance of Computers

- Want to buy the fastest computer for what you want to do
  - workload is important
- Want to design the fastest computer for what they want to pay
  - cost is an important criterion

Outline

- Time and performance
- Iron law
- MIPS and MFLOPS
- Which programs and how to average
- Amdahl's law

Defining Performance

- What is important to whom
1. Computer system user
   - minimize elapsed time for progra[m] time_end - time_start
   - called response time
2. Computer center manager
   - maximize completion rate = #jobs/second
   - called throughput

Limited model "Start job, wait for end." Suggest alternate performance metrics.

Response Time vs. Throughput

- Is throughput = 1/av. response time?
  - ONLY if NO overlap
  - with overlap, throughput > 1/av.response time
- e.g., a lunch buffet - assume 5 entrees
  - each person takes 2 minutes at every entree
    - throughput is 1 person every 2 minutes
    - BUT time to fill up tray is 10 minutes
  - Why? Otherwise, what would the throughput be?
    - because there are 5 people (each at 1 entree) simultaneously;
    - if there is no such overlap throughput = 1/10

What is Performance for us?

- For computer architects
  - CPU execution time = time spent running a program
- Shorter time better but people like better to be bigger to match intuition
  - performance = 1/time (shorter time better perf.)
  - where time = response time, CPU run time, etc.
- Elapsed time = CPU execution time + I/O wait
- We will concentrate mostly on CPU execution time

Improve Performance

- Improve (a) response time or (b) throughput?
  - faster CPU
    - both (a) and (b)
  - Add more CPUs
    - (b) but (a) may be improved due to reduced queueing

Give an example of this phenomenon

Performance Comparison

- Machine A is n times faster than machine B iff
  - perf(A)/perf(B) = time(B)/time(A) = n
- Machine A is x% faster than machine B iff
  - perf(A)/perf(B) = time(B)/time(A) = 1 + x/100
- E.g., A 10s, B 15s
  - 15/10 = 1.5 => A is 1.5 times faster than B
  - 15/10 = 1 + 50/100 => A is 50% faster than B

Breaking down Performance

- A program is broken into instructions
  - H/W is aware of instructions, not programs
- At lower level, H/W breaks instructions into cycles
  - lower level state machines change state every cycle
- E.g., 4 GHz Opteron runs 4 B cycles/sec, 1 cycle = 0.25 ns

Iron law

- Time/program = instrs/program x cycles/instr x sec/cycle
  - NEVER forget this!
- sec/cycle (a.k.a. cycle time, clock time)
  - mostly determined by technology and CPU orgn.
- cycles/instr (a.k.a. CPI)
  - mostly determined by ISA and CPU organization
  - EFFECTIVE cycles/instr and NOT actual latency
  - overlap among instructions makes this smaller
    - Each instr 5 cycles but 5 instrs overlap → CPI = 1
  - AVERAGE over instrs (instrs have different CPI)
- instr/program (a.k.a. instruction count)
  - instrs executed, NOT static code
  - mostly determined by program, compiler, ISA

Our Goal

- Minimize time which is the product, NOT isolated terms
  - Tempting because of the separate terms
  - Optimizing one term may worsen time by worsening other terms!
- Common error to miss terms while devising optimizations
  - E.g., ISA change to decrease instr. count
  - BUT leads to slower clock

Iron Law Example

- Machine A: clock 1 ns, CPI 2.0, for a program
- Machine B: clock 2 ns, CPI 1.2, for same program
- Which is faster and how much?
- Time/program = instrs/program x cycles/instr x sec/cycle
  - Time(A): N x 2.0 x 1 = 2N
  - Time(B): N x 1.2 x 2 = 2.4N
- Compare: Time(B)/Time(A) = 2.4N/2N = 1.2
- So, Machine A is 20% faster than Machine B for this program

Iron Law Example

- Keep clock of A at 1 ns and clock of B at 2 ns
- For equal performance, if CPI of B is 1.2, what is A's CPI?
  - Time(B)/Time(A) = 1 = (N x 2 x 1.2)/(N x 1 x CPI(A))
  - CPI(A) = 2.4

Iron Law Example

- Let CPI of A = 2.0 and CPI of B = 1.2
- For equal performance, if clock of B is 2 ns, what is A's clock?

Iron Law Example

- Let CPI of A = 2.0 and CPI of B = 1.2
- For equal performance, if clock of B is 2 ns, what is A's clock?
  - Time(B)/Time(A) = 1 = (N x 2.0 x clock(A))/(N x 1.2 x 2)
  - clock(A) = 1.2 ns

Iron Law notes

- You will see ch4a (single cycle) in the lab
- But not Ch1b (Iron law) so spend an hour to think about and internalize Ch1b

Other Metrics

- Other than execution time
- MIPS and MFLOPS
- MIPS = millions of instructions per sec

  = instruction count/(execution time x 10^6)

  - Not MIPS, the instruction set

  = clock rate/(CPI x 10^6) (How?)
- BUT MIPS has problems

Problems with MIPS

- E.g., without FP H/W, an FP op may take 50 single-cycle instrs
- with FP H/W only one 2-cycle instr at same clock
- Thus adding FP H/W
  - CPI increases (why?) The FP op goes from 50/50 to 2/1
  - but instrs/prog decreases more (why?) each of the FP op reduces from 50 to 1, factor of 50
  - total execution time decreases
- For MIPS
  - instrs/prog ignored
- MIPS gets worse!

Problems with MIPS

- Ignores program
- Usually used to quote peak performance
  - ideal conditions => guarantee not to exceed!!
- When is MIPS ok?
  - same compiler and same ISA
  - e.g., same binary running on Xeon Ivy Bridge and Xeon Broadwell
  - why? instrs/prog is constant and may be ignored

Other Metrics

- MFLOPS = FP ops in program/(execution time x 10^6)
- Assuming FP ops independent of compiler and ISA
  - Assumption not true
  - Eg may not have divide instruction in ISA
  - optimizing compilers can remove some insts
- Relative MIPS and normalized MFLOPS
  - adds to confusion!

Rules

- Use ONLY Time
  - Beware when reading, especially if details are omitted
  - Beware of Peak
    - Peak is the performance that the chip maker guarantees not to exceed – meaningless!

Outline

- Time and performance
- Iron law
- MIPS and MFLOPS
- Which programs
- How to average
- Amdahl's law

Which Programs

- Execution time of what
- Best case - you run the same set of programs everyday
  - port them and time the whole "workload"
- In reality, use benchmarks
  - programs chosen to measure performance
  - predict performance of actual workload (hopefully)
  - saves effort and money
  - representative? honest?

Benchmarks: SPEC2006

- SPEC: System Performance Evaluation Cooperative
- Latest is SPEC2006, before SPEC89, SPEC92, SPEC95, SPEC2000
- 12 integer and 17 floating point programs
  - normalize run time with a Gold Standard processor (*SPEC ratio*)
  - Geometric Mean (GM) of the normalized times (why GM?)

SPEC CINT2006
http://www.spec.org/cpu2006/CINT2006/

Benchmark	Description
Xalancbmk	XML processing
astar	Path finding
mcf	Combinatorial optimization
gcc	GNU C compiler
Omnetpp	Discrete event simulation
h264ref	Video compression
libquantum	Quantum computing
gobmk	Artificial Intelligence: Go
hmmer	Gene Sequence Search
Bzip2	Compression
sjeng	Artificial Intelligence: chess
Perlbench	PERL programming language

SPEC CFP2006
http://www.spec.org/cpu2006/CFP2006/

Benchmark	Description
milc	Quantum chromodynamics
Bwaves	Fluid dynamics
Soplex	Simplex LP solver
Wrf	Weather modeling
17 in all, see webpage	Speech recognition, quantum chemistry, structural mechanics, ray tracing, etc.

How to average

- SPEC has 29 programs – how to average?
- Example

	Machine A	Machine B
Program 1	1s	10s
Program 2	1000s	100s
Total	1001s	110s

- One answer: total execution time, then how much faster than A is B? 1001/110 = 9.1

How to average

- Another: arithmetic mean (same result: B 9.1 times faster than A)
- Arithmetic mean of times: $\left\{\sum_{i=1}^{n} time_i\right\} / n$ for n programs
- AM(A) = 1001/2 = 500.5
- AM(B) = 110/2 = 55
- 500.5/55 = 9.1
- Valid only if programs run equally often, else use "weight" factors
- Weighted arithmetic mean: $\left\{\sum_{i=1}^{n} weight_i \times time_i\right\} / n$

Other Averages

- E.g., 30 mph for first 10 miles
- 90 mph for next 10 miles. Average speed?
- Average speed = (30+90)/2 =60mph? WRONG
- Average speed = total distance / total time
  = (20 / (10/30+10/90))
  = 45 mph
- What if it was 10 hours at each speed?
  - instead of 10 miles

Harmonic Mean

- Harmonic mean of rates = $\dfrac{1}{\left\{\sum_{i=1}^{n} \dfrac{1}{rate_i}\right\} / n}$
  - Use HM if forced to start and end with rates
- Trick to do arithmetic mean of times but using rates and not times

Practice

- Machine A runs 10M instructions at 15 MIPS and the next 10M instructions at 45 MIPS
  - Average MIPS = ??
- Machine A runs for 10 seconds at 15 MIPS and the next 10 seconds at 45 MIPS
  - Average MIPS = ??

Dealing with Ratios

- Absolute times

	Machine A	Machine B
Program 1	1s	10s
Program 2	1000s	100s

- Now consider ratios (w.r.t. A)

	Machine A	Machine B
Program 1	1	10
Program 2	1	0.1

- Averages: A = 1, B = 5.05

Dealing with Ratios

- Absolute times

	Machine A	Machine B
Program 1	1s	10s
Program 2	1000s	100s

- Now consider ratios (w.r.t. B)

	Machine A	Machine B
Program 1	0.1	1
Program 2	10	1

- Averages: A = 5.05, B = 1
- Both cannot be true

Geometric Mean

- Don't use arithmetic mean on ratios (normalized numbers)
- Use geometric mean for ratios
  - geometric mean of ratios = $\sqrt[n]{\prod_{i=1}^{n} ratio_i}$
  - Use GM if forced to use ratios
- Independent of reference machine (math property)
- In the example, GM for machine A is 1, for machine B is also 1
- normalized with respect to either machine

But..

- Geometric mean of ratios is not proportional to total time
- AM in example says machine B is 9.1 times faster
- GM says they are equal
- If we took total execution time, A and B are equal only if
  - program 1 is run 100 times more often than program 2
- Generally, GM will mispredict for three or more machines

Previous Midterm Question

- Machine A runs 9 times faster than machine B when running program-P,
- Machine B runs 4 times faster than machine A when running program Q.

- Which machine is faster (and by what factor) when averaged across both programs?

Summary for Averages

- Use AM for times
- Use HM if forced to use rates
- Use GM if forced to use ratios
- Better yet, use unnormalized, raw run times to compute performance

Amdahl's Law

- Why does the common case matter the most?
- Let an optimization speed f fraction of time by a factor of s
- assuming that old time = T, what is the speedup?
  - f is the "affected" fraction of T
  - (1-f) is the unaffected fraction
- Speedup = $\frac{time_{old}}{time_{new}} = \frac{unaffected_{old} + affected_{old}}{unaffected_{new} + affected_{new}}$
- = $\frac{(1-f) \times T + f \times T}{(1-f) \times T + \frac{f}{s} \times T} = \frac{1}{(1-f) + \frac{f}{s}}$

Amdahl's Law Example

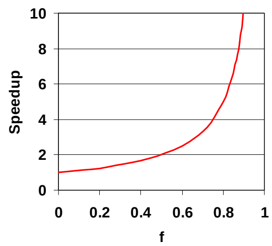
- Your boss asks you to improve processor performance
- Two options: What should you do?
  - improve the ALU used 95% of time, by 10%
  - improve the square-root unit used 5%, by a factor of 10

f	s	Speedup
95%	1.10	1.094
5%	10	1.047
5%	∞	1.052

Amdahl's Law: Limit

- Make common case fast because:

$$\lim_{s\to\infty}\left(\frac{1}{1-f+f/s}\right)=\frac{1}{1-f}$$



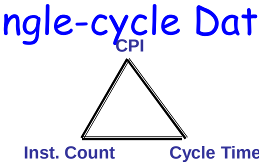
Amdahl's Law

- "Make common case fast"
  - Scientific heuristic, not religious commandment
  - Use for intuition, verify with numbers
- 60% can be improved by a factor of 2
  - Speedup = 1/(0.4+0.6/2) = 1/0.7
- 40% can be improved by a factor of 8
  - Speedup = 1/(0.6+0.4/8) = 1/0.65
- Second option is better
  - Less common case, but higher speedup compensates

Summary of Chapter 1b

- Time and performance:
  - Machine A n times faster than Machine B
  - iff Time(B)/Time(A) = n
- Iron Law: Time/prog
  - Instr count x CPI x Cycle time
- Other Metrics: MIPS and MFLOPS
  - Beware of peak and omitted details
- Benchmarks: SPEC2006 (int + FP)
- Summarize performance:
  - AM for time, HM for rate, GM for ratio
- Amdahl's Law: Speedup = $\left(\frac{1}{1-f+f/s}\right)$ common case fast

Single-cycle Datapath



- Performance Implications
  - Minimize all three
  - Insts/prog fixed -- f(interface,compiler)
  - CPI = 1 : As good as it gets (*)
  - Clock cycle time : high, "lw" critical path

Back to Ch4b

- To improve performance beyond single-cycle datapath