Quantitative Analysis

Today's topics:

- failure analysis
- performance analysis
- some basic quantitative principles
- caution – pot holes – it's easy to lie w/ numbers

Some Issues So Far

- And it's only the 2nd Class
- You'll note my preference
  - conceptual stuff in the lectures
  - practical stuff in the homeworks
    - » give me feedback when this approach isn't good enough
- Text isn't in the bookstore
  - major screw-up
    - » due to late teaching assignment change
    - » order it on-line
      - It'll be faster and cheaper
- Homework #1 will be on the web later today
  - make sure you start early
  - holiday weekend ahead
    - » maybe you'd like to enjoy it

Reliability

- Reliability is a key concern in some segments
  - mission critical embedded systems
    - » e.g. nuclear power plants, automotive, aero & space, …
  - when high availability is needed
    - » either due to monetary loss or contract
      - SLA's and SLO's
- Weakest link theory
  - useful acronyms (note these are averages & "user mileage may vary")
    - » MTTF – mean time to failure
    - » MTTR – mean time to repair
    - » MTBF (B=between) = MTTF + MTTR
    - » availability = MTTF/MTBF
  - hook?
    - » simple for a module – more complex for a larger system

Failure Mechanisms

- 2 types
  - hard – permanent failure
  - transient – temporary failure
    - » due to environmental issues
      - alpha particles, heat, cross-talk, noise, vibration, …
- Device specific (small set of examples)
  - IC's
    - » transistors can fail due to excess heat & current
      - extremely reliable in general
    - » wires fail due to excess current – "metal migration"
  - Disks (checkout recent Google paper on this)
    - » MHD's: oxide deterioration, head saturation, coil-motor accuracy
    - » SSD's: block erase oxide thinning
  - DRAM's (checkout recent Google paper on this too!)
    - » IC's but alpha particles disrupt stored charge

Improving Reliability

- 2 strategies
  - build more reliable devices
    - » more costly & a very slippery slope
  - use more of them → redundancy
- Redundancy shows up in lots of costumes
  - extra bits – CRC & ECC codes
    - » even more exotic: Turbo, Viterbi, etc.
  - extra gates and wires
    - » seldom used today
  - redundant blocks
    - » 2: compare and signal error if they don't agree
    - » some odd number: vote and take majority, flag anyway
  - redundant everything
    - » retry elsewhere if something fails
  - hybrid
    - » e.g. NAND Flash – ECC on block, quarantine block before things get nasty

Performance

- 2 aspects
  - throughput: rate of completion of multiple jobs, processes, or threads
  - single thread performance or *execution time*
  - making one better usually degrades the other
- Comparing: performance = 1/execution_time

$$\frac{\text{Execution Time}_Y}{\text{Execution Time}_X} = N$$

  - similar game for throughput comparisons

Measuring Performance

- Tricky in today's multiprocessing world
  - alias factors
    - » elapsed time (stopwatch) is load dependent
    - » context switch
      - process is swapped out part of the time it's supposedly running
    - » page faults
      - only fair if your workload is the only one running
    - » I/O delays
      - processing may be dwarfed by slow I/O response time
    - » OS overheads
      - fair if OS service is important part of your workload
      - unfair if service to other workloads are observed
- Fortunately
  - tools exist to help break out time into different bins
    - » still some cruft gets swept under the rug

Tools

- Unix time command
  - otb> time
    - » 0.898u 0.311s 2:39.79 0.7% 0+0k 0+0io 9pf+02
  - meaning
    - » u = seconds of user process execution time
    - » s = seconds of system execution time (OS)
    - » 2:39.79 minutes of elapsed time
      - includes page faults, I/O overhead, etc. (a.k.a. external overheads)
    - » k = KB of text + data used
    - » io = amount of I/o sent
    - » pf: major plus minor page faults
      - major: page was on disk
      - minor: TLB miss but page in main-memory (DRAM)
  - Beware: OS "system time" undervalued
    - » call and return linkages usually charged to user time
- Higher fidelity
  - use on chip counters via some tool like Intel's vTune

Lots of Performance Analysis Tools

- Key is to learn what they're good at
  - some are good at
    - » tracking certain HW events – cache misses, TLB misses, IPC
    - » course grained phase changes
      - aggregate finer details into a larger "average"
- Point
  - use the right tool for the job
  - seems obvious but often users don't get it
- Some things are very hard
  - each tool has a *"probe effect"*
    - » often hard to determine the overhead
      - partially because it may be inconsistent

Evaluating Machines

- Which programs do you choose?
  - real programs
    - » ideal but problematic
      - you can't just read about them
      - it's a lot of work
      - what you care about may be diverse and change over time
  - kernels
    - » computationally intensive pieces of your programs
      - same problem as above PLUS
        - you have to profile your code to find the right stuff
        - intuition of where the time goes is suspect
      - use existing kernels
        - e.g. Livermore Loops & Linpack
          - small loops over big data sets
        - good chance they don't represent your computational needs
        - not real programs anyway
          - just stress the CPU
- What would you do?
  - without looking at the next slide!

Benchmarks

- Industry standard reporting mechanism
  - burden
    - » need to understand what the benchmark measures
      - int, float, cache, main-memory, interconnect, ...
    - » enormous diversity in today's benchmarks
- Common benchmark suites
  - SPEC: http://www.spec.org/benchmarks.html
    - » standard set for desktop/laptop segment
      - both int and fp codes
    - » extensions: OpenMP, MPI, graphics
  - PARSEC: http://parsec.cs.princeton.edu/overview.htm
    - » new suite aimed at multi-core processor evaluation
  - EEMBC: http://eembc.org/benchmark/index.php
    - » diverse suite aimed at embedded systems
      - telecom, automotive, networking, multicore, ...

More Benchmarks

- TPC: http://www.tcp.org
  - transaction processing servers (like Google)
    - » heavy on I/O – somewhat light on processing
    - » examples
      - TPC-A: simple bank teller transactions
      - TPC-C: compex database query
        - heavy memory and disk usage
      - TPC-H: decision support
        - lots of data but what does it mean
      - TPC-W: web server

Benchmark Issues

- Reproducibility
  - a must – hence test jig is specified
- Source code modifications
  - SPEC – not allowed
  - TPC – allowed but too difficult to be probable
  - Linpack & Livermore Loops – allowed
  - EEMBC – even allows hand assembly coding
- Various cheating mechanisms
  - compiler recognizes benchmark and emits hand coded .asm
  - allow programming practice to evolve
    - » particularly true for newer architectures
      - e.g. multi-(thread/core/...)
  - tough line to walk
- Key (worth repeating)
  - Know what each benchmark is really measuring

Trusting Reported Performance

- Depends – initially be skeptical
  - need
    - » precise machine configuration and test setup
  - things are actually pretty good today
  - some venues are better than others
    - » Microprocessor Forum – highly reliable
    - » Internet – it's a crap shoot
    - » popular press
      - key is to figure out what their source is

Usually care about more than 1 program

- Example

	Machine A	Machine B	Machine C
Program 1 (secs)	1	10	20
Program 2 (secs)	1000	100	20
Total Time (secs)	1001	110	40

Which is better and by how much?

Aggregation Options

- Arithmetic Mean
  - Arithmetic Mean – simple average

$$\frac{1}{n}\sum_{i=1}^{n} \text{Time}i$$

    - doesn't account for weight/importance
  - Weighted AM

$$\sum_{i=1}^{n} \text{Weight}i \times \text{Time}i$$

    - better but beware the dominant program time

Using Rate/Throughput

- **Harmonic Mean**
  - Harmonic mean

$$\frac{n}{\sum_{i=1}^{n} \frac{1}{\text{Rate}i}}$$

  - Weighted HM

$$\frac{n}{\sum_{i=1}^{n} \frac{\text{Weight}i}{\text{Rate}i}}$$

Dealing with large time variations

- **Geometric Mean**
  - Geometric Mean

$$\sqrt[n]{\prod_{i=1}^{n} \text{Execution Time Ratio}i}$$

  - Properties
    - ratio of the means = mean of the ratios
    - no dominance of longest run time in the result
    - probably better than AM or HM
    - weighted variant also possible
      - » probably the best of all IF you can assign the weights properly

General Principles

- **Make the common case fast**
  - you just need to figure out what it is
  - easy to say hard to do
    - » HW → fast, SW →slow, hence
      - HW support for common case but it's inflexible
      - SW support for flexibility
  - some issues are simple
    - » exceptions are rare → handle it software
      - but recognize it w/ HW support
- **Whatever you do it has to work**
  - reliably and within a cost parameter that the market will bear
    - » note academics often are unconcerned with this
      - short lived companies may be in the same boat
    - » for industry
      - everything matters since you have to build the whole system

Amdahl's Law

- Speedup of a particular feature
  - XEQ-time = 1/performance so other variants are possible

$$\text{Speedup} = \frac{\text{Execution time without using the enhancement}}{\text{Execution time using the enhancement}}$$

- Amdahl's law
  - quantifying the commonality factor

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

  - beware: Amdahl's law says nothing about cost

Simple Example

- 3 instructions considered key enhancements for graphics
  - FP instructions (except FPSQRT) 50% of dynamic count
  - FPSQRT 20%
  - all other instructions 30%
- Designers say: "for the same cost/area we can speed up
  - FP by 2x
  - FPSQRT by 40x
  - all others by 8x
- Trick – in this case you only get to pick one
  - what's your guess
    - » numbers next but what is your intuition?

Answer

- FPSQRT

$$\frac{1}{(1-\text{Fraction}_{\text{enhanced}})+\frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}} = \text{Speedup}_{\text{FPSQRT}} = \frac{1}{(1-0.2)+\frac{0.2}{40}} = 1.242$$

- FP

$$\text{Speedup}_{\text{FP}} = \frac{1}{(1-0.5)+\frac{0.5}{2}} = 1.333$$

- Other

$$\text{Speedup}_{\text{Other}} = \frac{1}{(1-0.3)+\frac{0.3}{8}} = 1.356$$

Biggest enhancement is the loser as is the most frequent!!

Calculating Performance

- Simple view

$$\text{CPU time} = \text{CPU cycles for a program} \times \text{Clock Cycle Time}$$

$$\text{CPU time} = \frac{\text{CPU cycles for a program}}{\text{Clock Rate}}$$

  - what's wrong with this?
    - » frequency = 1/cycle-time

Easier to count instructions

- Dynamic count is what you need
  - static count → footprint
    - » not a big deal these days except in the embedded segment
- Problem
  - not all instructions take the same number of cycles
    - » we'll see why later
    - » e.g. FPDIV is way more work than a shift-left 4-bits
- 2 new terms
  - IC = dynamic instruction count
  - CPI = cycles per instruction
    - » today IPC is used due to multi-issue architectures
      - IPC = 1/CPI
- For a given workload
  - IPC is a figure of merit

$$\text{CPU time} = \text{IC} \times \text{CPI} \times \text{Cycle Time} = \frac{\text{IC} \times \text{CPI}}{\text{Clock Rate}}$$

IC, IPC, Cycle-Time Influences

- IC
  - depends on the instruction set & compiler
    - » ADD vs. DFT as a silly example
- IPC
  - depends on ISA and machine architecture
- Frequency
  - depends on the pipeline depth
    - » more soon on this but too deep has it's issues
    - » too shallow →too little parallelism
- Conflicting constraints
  - Improving on one is easy
    - » without making the others worse is hard

Other Factors

- It's not all about performance
  - even though a lot of our focus in 6810 is centered here
- Cost
  - performance/$
    - » see some examples in your text
  - TCO – total cost of ownership
    - » e.g. how reliable and how long does it last
  - upgrades, SW, peripherals, …, long list
    - » processor may be a small piece of the whole system
  - ease of use if your time is worth anything
- Power
  - important when you're not plugged in
  - too hot → more expensive cooling required
    - » so power costs more than just on your electric bill