18-447 Lecture 7: Performance -- how to summarize & compare

James C. Hoe
Dept of ECE, CMU
February 9, 2009

Announcements: Midterm 2/16 in class, Lectures 1~7
Read P&H Ch 5 for next Lecture

Handouts: MIPS R4000 ISA Manual on BlackBoard

Latency vs. Throughput

- Latency (a time measure)
  - time between start and finish of a single task
  - most applicable in interactive applications
- Throughput (a rate measure)
  - number of tasks finished in a given unit of time
  - most applicable in batch applications
- Throughput is not always 1/latency when concurrency is involved (think bus vs. F1 race car)
  - improve latency $\Rightarrow$?? improve throughput
  - improve throughput $\Rightarrow$?? improve latency
- Not completely distinct when different granularities are considered
  - increasing throughput of component processing shortens the latency of the overall task

It is all about time

- Performance = 1 / Time
  - shorter latency ⇒ higher performance
  - higher throughput (job/time) ⇒ higher performance
- UNIX "time" command
  - user CPU time: time spent running your code
  - system CPU time: time spent running other code on behalf of your code
  - elapsed time: wall-clock time
  - elapsed time – user CPU time – system CPU time = time running other code unrelated to your code

1. Be precise about what you measured when reporting
2. Rule of thumb: measure and report wall-clock time on unloaded system

IPC, MIPS and GHz

- The metrics you are most likely to see in marketing are IPC (instruction per cycle), MIPS (million instruction per second) and GHz

  How are they incomplete?
- Iron Law on Performance

wall clock time = (time/cyc) (cyc/inst) (inst/program)

- MIPS and IPC are averages — which instructions matter
- GHz can be boosted artificially by design (lower the other 2 terms) e.g., 1.4GHz P4 ≈ 1.0GHz P3

Pseudo FLOPS

- Scientific computing community often use pseudo FLOPS as performance metric

$$\frac{\text{nominal \# of floating point operations}}{\text{program runtime}}$$

  - e.g. FFT of size N has nominally $5N \log_2(N)$ FP operations

- Is this a good, fair metric to compare machine + algorithm combinations?
  - not all FFT algorithms have the same FP OP count
  - not all FP OPs are equal (FADD vs FMULT vs FDIV)

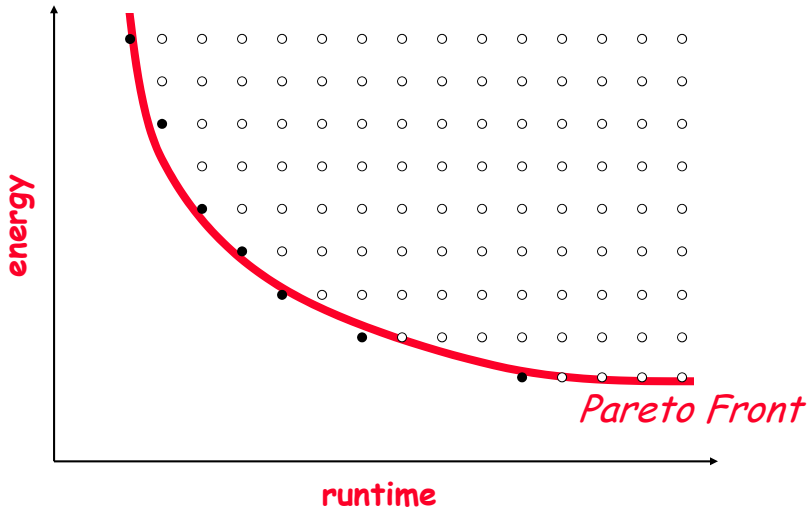
Ans: yes, but only as long as you are talking about computing the same problem

Multi-dimensional Comparisons: e.g., Runtime and Energy

- Interested in not only minimizing individual metrics but also consider the tradeoff between them, i.e.,
  - may be willing to spend more energy to run faster
  - conversely, may be willing to run slower for less energy spent
  - but never use more energy to run slower
- Derived combined metrics of interest
  - power = E / T
  - energy-delay-product = $E \cdot T$
  - in general, **f**(E, T)

Other dimensions: implementation cost, risk, social factors…

Pareto Optimality



Comparing and Summarizing Performance

Relative Performance

- Performance = 1 / Time
  - shorter latency ⇒ higher performance
  - higher throughput (job/time) ⇒ higher performance
- Pop Quiz

  if X is 50% slower than Y and $Time_X$=1.0s, what is $Time_y$

  - Case 1: $Time_y$ = 0.5s since $Time_y/Time_X$=0.5
  - Case 2: $Time_y$ = 0.66666s since $Time_X/Time_y$=1.5

Relative Performance

- "X is n times faster than Y" means

  $n = Performance_X / Performance_y$

  $= Throughput_X / Throughput_y$

  $= Time_y / Time_X$
- "X is m% faster than Y" means

  $1+m/100 = Performance_X / Performance_y$
- To avoid confusion, stick with definition of "faster"
  - for case 1 say "Y is 100% faster than X"
  - for case 2 say "Y is 50% faster than X"

Speedup

- If X is an "enhanced" version of Y, the "speedup" of the enhancement is

$$S = Time_{without\ enhancement} / Time_{with\ enhancement}$$
$$= Time_Y / Time_X$$

Amdahl's Law on Speedup

- Suppose an enhancement speeds up a fraction f of a task by a factor of S_f

$$time_{new} = time_{old} \cdot ((1-f) + f/S_f)$$
$$S_{overall} = 1 / ((1-f) + f/S_f)$$

If f is small S_f doesn't matter. Concentrate effort on improving frequently occurring events or frequently used mechanisms.

Summarizing Performance

- When comparing two computers X and Y, the relative performance of X and Y depends strongly on what X and Y are asked to do
  - X may be m% faster than Y on application A
  - X may be n% (where m!=n) faster than Y on application B
  - Y may be k% faster than X on application C
- Which computer is faster and by how much?
  - depends on which application(s) you care about
  - if you care about several applications, then it also depends their relative importance
- Many ways to summarize performance comparison into a single quantitative measure
  - some may even be meaningful for exactly your purpose
  - but you have to know when to do what
  - when in doubt, present the complete story

Arithmetic Mean

- Suppose you workload is applications $A_0, A_1, .. A_{n-1}$
- Arithmetic mean of the application run time is

$$\frac{1}{n}\sum_{i=0}^{n-1} Time_{A_i}$$

  - comparing AM is the same as comparing total run-time
  - caveat: longer applications have greater contribution than shorter applications
- If AM_X/AM_Y=n then Y is n times faster than X

  True: $A_0,...A_{n-1}$ are run equal number of times always

  False: if some applications are run much more frequently then others *(especially problematic if the most frequent applications are also much shorter than the rest)*

Weighted Arithmetic Mean

- Introduce weighting factors, $w_0, w_1 \ldots w_{n-1}$ where $1 = \sum_{i=0}^{n-1} w_i$
- w_i is the number of times A_i runs relative to total number of times any program in the workload is run
- Weighted arithmetic mean of the run time is

$$\sum_{i=0}^{n-1} w_i \cdot Time_{A_i}$$

- If $WAM_X/WAM_Y=n$ then Y is n times faster than X on a workload characterized by $w_0, w_1 \ldots w_{n-1}$
- But w_i isn't fixed and isn't easy to come by, how about $\frac{1}{n}\sum_{i=0}^{n-1} \frac{Time_{A_i \text{ on } X}}{Time_{A_i \text{ on } Y}}$ or $\sqrt[n]{\prod_{i=0}^{n-1} \frac{Time_{A_i \text{ on } X}}{Time_{A_i \text{ on } Y}}}$

Yes, you get a number at the end, but what does it mean?

Normalized Performance

- Suppose
  - A_0 takes 1s on X; 10s on Y; and 20s on Z
  - A_1 takes 1000s on X; 100s on Y; and 20s on Z
  - A_0+A_1 = 1001s on X; 110s on Y; and 40s on Z

	normalized to X			normalized to Y			normalized to Z		
	X	Y	Z	X	Y	Z	X	Y	Z
$Time_{A0}$	1	10	20	0.1	1	2	0.05	0.5	1
$Time_{A1}$	1	0.1	0.02	10	1	0.2	50	5	1
AM of ratio	1	5.05	10.01	5.05	1	1.1	25.03	2.75	1
GM of ratio	1	1.0	0.63	1.0	1	0.63	1.58	1.58	1

[Computer Architecture: A quantitative approach. Hennessy and Patterson]

Harmonic Mean

- Don't take arithmetic mean of rates (e.g. throughput)
  - e.g. 30 mph for first 10 miles, 90 mph for next 10 miles, the average speed is not (30 + 90)/2 = 60 mph!
- To compute average rate
  1. expand fully

     average speed = total distance / total time

     = 20 / (10/30 + 10/90) = 45 mph
  2. harmonic mean

$$HM = n \Big/ \sum_{i=0}^{n-1} \frac{1}{Rate_i} \qquad WHM = 1 \Big/ \sum_{i=0}^{n-1} \frac{w_i}{Rate_i}$$

HM is just a short-cut for doing the fully expanded calculation when averaging rates

Standard Benchmarks

- Why standard benchmarks?
  - Everyone cares about different applications (different aspects of performance)
  - Your application may not be available for the machine you want to study
- SPEC Benchmarks (www.spec.org)
  - Standard Performance Evaluation Corporation
  - a set of "realistic", general-purpose, public-domain applications chosen by a multi-industry committee
  - updated every few year to reflect changes in usage and technology
  - a sense of objectivity and predictive power
  - everyone knows it is not perfect, but at least everyone plays/cheats by the same rules

Other similar industry entities also exist for other application domains (server, embedded, etc)

SPEC CPU Benchmark Suites

(http://www.spec.org/cpu2006)

- CINT2006 (C unless otherwise noted)

  perlbench (prog lang), bzip2 (compress), gcc (compile),mcf (optimize), gobmk (go), hmmer (gene seq. search), sjeng (chess), libquantum (physics sim.), h264ref (video compress), omnetpp (C++, discrete event sim.), astar (C++, path-finding), xalancbmk (C++, XML)

- CFP2006 (F77/F90 unless otherwise noted)

  bwaves (CFD), gamess (quantum chem), milc (C, QCD), zeusmp (CFD), gromacs (C+Fortran, molecular dyn), cactusADM (C+Fortran, relativity), leslie3d (CFD), namd (C++, molecular dyn), dealII (C++, finite element), soplex (C++, Linear Programming), povray (C++, Ray-trace), calculix (C+Fortran, Finite element), GemsFDTD (E&M), tonto (quantum chem), lbm (C, CFD), wrf (C+Fortran, weather), sphinx3 (C, speech recog)

- Reports geometric mean of normalized performance relative to a 296MHz reference Sun UltraSparc II

Performance Summary

- There is no one-size-fits-all methodology
  - be sure you understand what you want to measure
  - be sure you understand what you measured
  - be sure what you report is accurate and representative
  - be ready to come clean with raw data
- No one believes your numbers anyway
  - be clear about what effect you are trying to measure
  - be clear about what and how you actually measured
  - be clear about how performance is summarized and represented

If the results really matter, people will want to check it for themselves