18-447 Lecture 5: Performance and All That (Uniprocessor)

James C. Hoe
Department of ECE
Carnegie Mellon University

Housekeeping

- Your goal today
  - appreciate the subtleties of measuring/summarizing/comparing performance
  - focus is on sequential execution performance
    - L12: power&energy; L23: parallel performance
- Notices
  - Lab 1, Part A, due this week
  - Lab 1, Part B, due next week
  - HW1, due Monday 2/5
- Readings
  - P&H Ch 1.6~1.9
  - P&H Appendix C for next time

It's about time

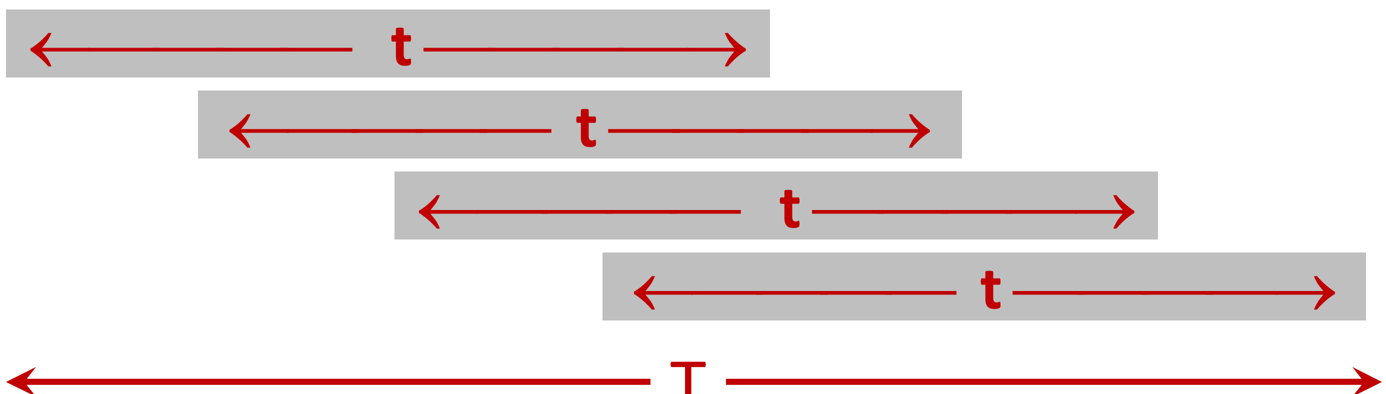
- To the first order, **performance $\propto$ 1 / time**

- Two very different kinds of performance!!
  - latency = time between start and finish of a task
  - throughput = number of tasks finished in a given amount of time (a rate measure)

- Either way, shorter the time, higher the performance, but not to be mixed up

Throughput $\neq$ 1/Latency : Littles' Law

- If it takes T sec to do N tasks, throughput=N/T; latency$_1$=T/N?
- If it takes t sec to do 1 task, latency$_1$=t; throughput=1/t?
- When there is concurrency, throughput$\neq$1/latency



- Optimizations can tradeoff one for the other

(think bus vs F1 race car)

Little's Law

- $L = \lambda \cdot W$
  - **L**: number of customers
  - λ: arrival rate
  - **W**: wait time

In 447 language:

 # overlapped ops

throughput

latency

- In steadystate, fix any two, the third is decided

- HW system examples
  - in-order instruction pipeline: ILP and RAW hazard distance determine instruction throughput
  - AXI DRAM read: latency and # outstanding requests determine achieved BW (until peak)

Throughput ≠ Throughput : Overhead Amortization

- Example: using DMA to transfer on a bus
  - bus throughput_{raw} = 1 Byte / (10^{-9} sec) *steadystate*
  - 10^{-6} sec to setup a DMA
  - $\text{throuhgput}_{effective}$ to send 1B, 1KB, 1MB, 1GB?

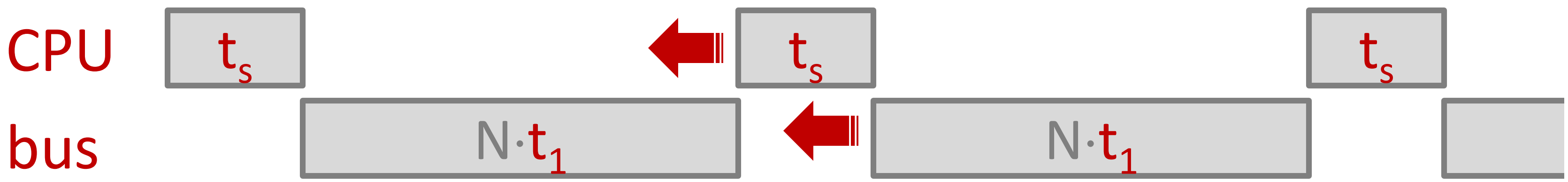
Throughput a function of transfer size due to non-recurring start-up cost (aka overhead)

- For start-up-time=t_s and $\text{throughput}_{raw}=1/t_1$
  - $\text{throughput}_{effective} = N / (t_s + N\cdot t_1)$
  - if $t_s >> N\cdot t_1$, $\text{throughput}_{effective} \approx N/t_s$
  - if $t_s << N\cdot t_1$, $\text{throughput}_{effective} \approx 1/t_1$

we say t_s is "amortized" in the latter case

Latency ≠ Latency : Latency Hiding

- What are you doing during the latency period?
- Latency = hands-on time + hands-off time
- In the DMA example
  - CPU is busy for the t_s to program the DMA engine
  - CPU has to wait $N \cdot t_1$ for DMA to complete
  - CPU could be doing something else during $N \cdot t_1$ to "hide" that latency



Sounds Like Performance

- The metrics you are most likely to see in microprocessor marketing
  - GHz (billion cycles per second)
  - IPC (instruction per cycle)
  - MIPS (million instructions per second)
- Incomplete and/or misleading
  - GHz and IPC have wrong units (not work/time)
  - MIPS and IPC are averages (depend on inst mix)
  - GHz, MIPS or IPC can be improved at the expense of each other and actual performance

e.g., 1.4GHz Intel P4 ≈ 1.0GHz Intel P3?

Iron Law of Processor Performance

- time/program = (inst/program) (cyc/inst) (time/cyc)

1/IPC 1/MIPS 1/GHz

note workload dependence

- Contributing factors
  - time/cyc: architecture and implementation
  - cyc/inst: architecture, implementation, instruction mix
  - inst/program: architecture, nature and quality of prgm

- **Note**: cyc/inst is a workload average

potentially large instantaneous variations
due to instruction type and sequence

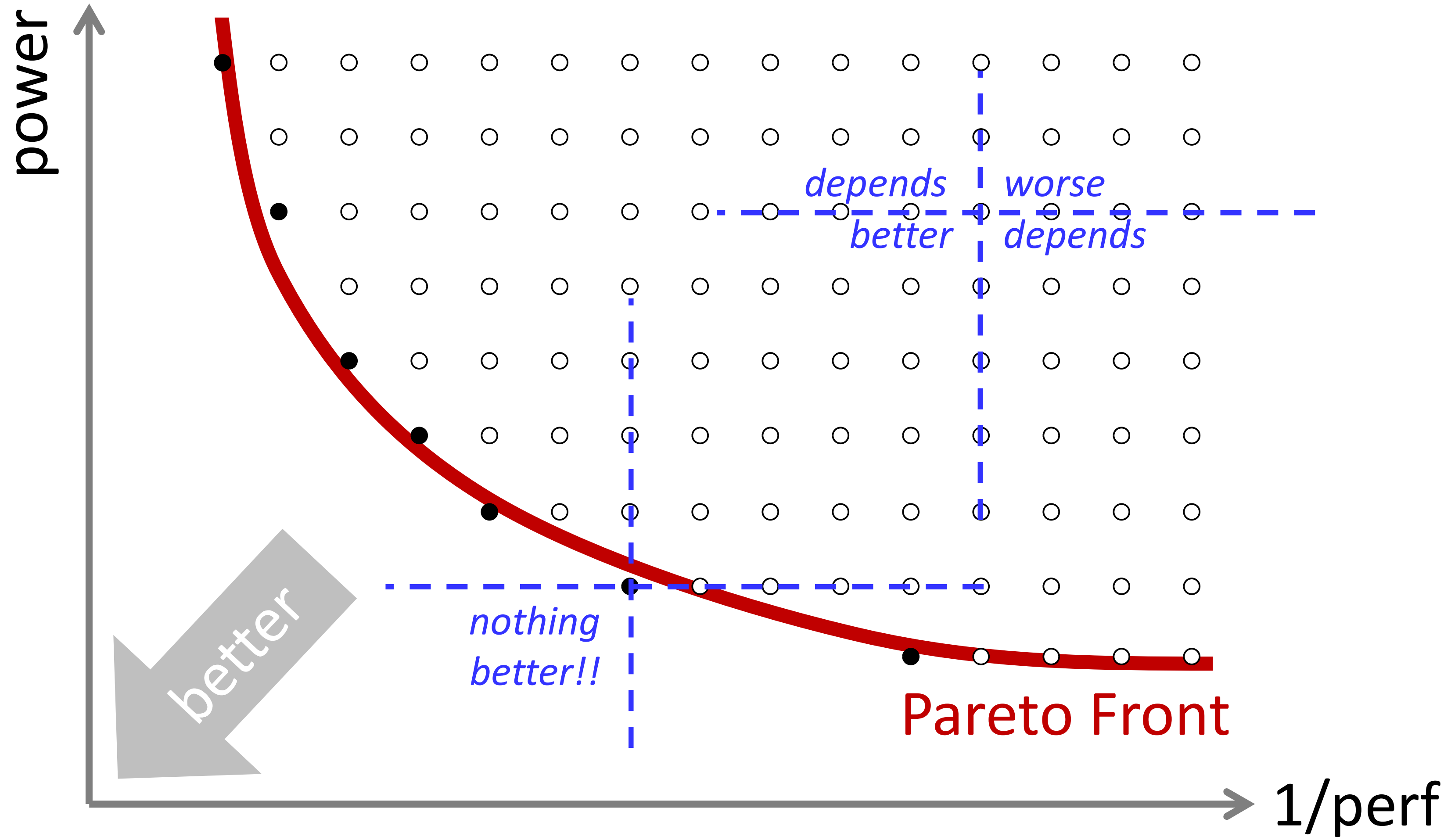
When it is about more than time

Tradeoff

- Other metrics of goodness beside "getting the right answer": performance, power/energy, cost, risk, social factors . . . ethics . . .
- Cannot optimize individual metrics without considering tradeoff between them
- E.g. runtime vs. energy
  - may be willing to spend more energy per task to run faster
  - conversely, may be willing to run slower for less energy per task
  - but never use more energy to run slower

"…$5.8 million the value of a statistical life…" FAA

Pareto Optimality (2D example)



All points on front are optimal (can't do better)

How to select between them?

Composite Metrics

- Define scalar function to reflect desiderata---incorporate dimensions and their relationships
- E.g., energy-delay product
  - can’t cheat by minimizing one ignoring other
  - but is smaller really better?

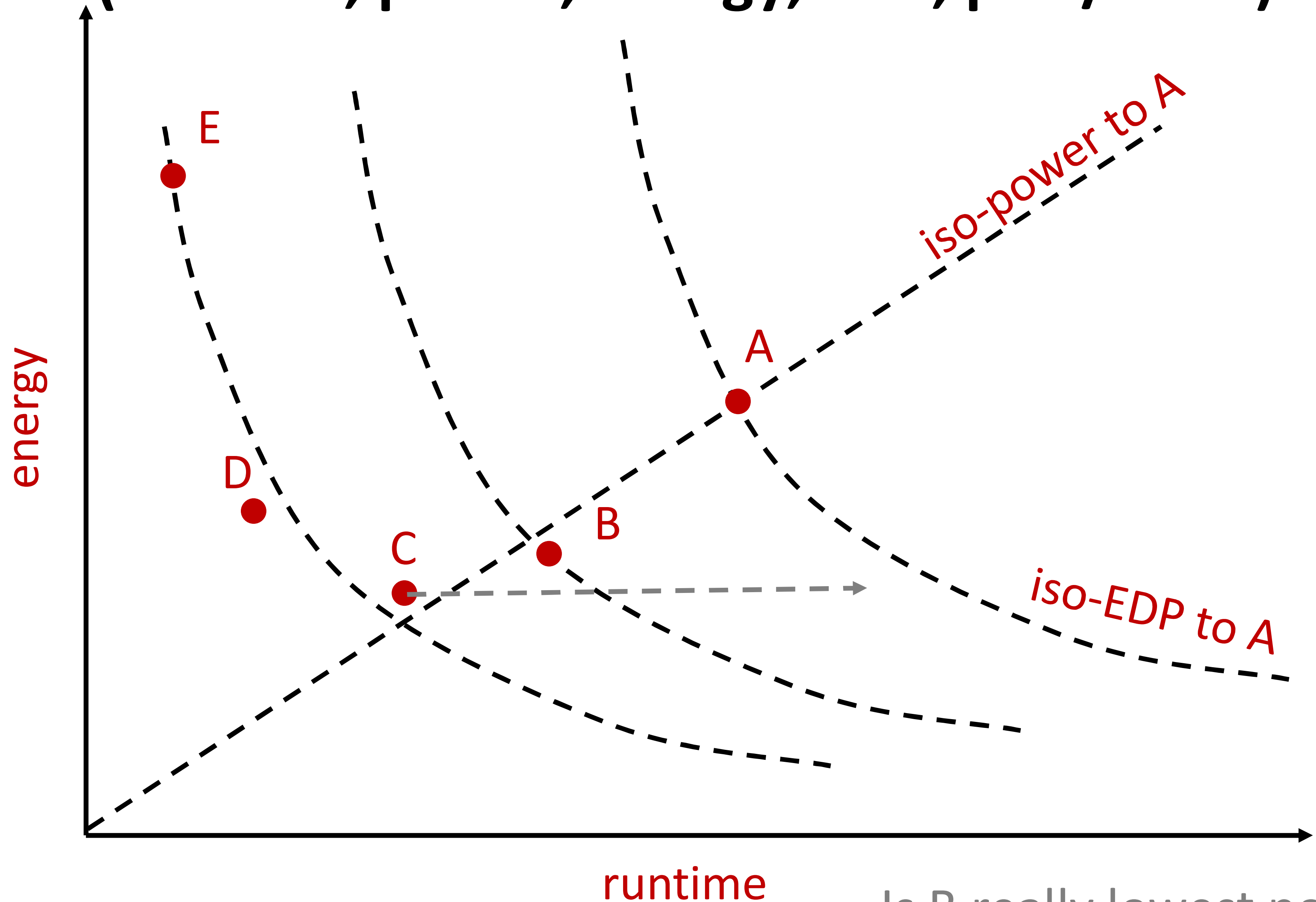
be wary of relevance to application context

- Floors and ceilings
  - real-life designs more often about good enough than optimal
  - e.g., meet a perf floor under a power(cost)-ceiling

Not all desires reducible to quantifiable terms!!

Which Design Point is Best?

(runtime, power, energy, EDP, perf/Watt)



Is B really lowest power?

Scale Makes a Difference in Normalization

- Perf/Watt and op/J are normalized measures
  - hides the scale of problem and platform
  - recall, Watt $\propto$ perf^k for some k>1
- 10 GFLOPS/Watt at 1W is a very different design challenge than at 1KW or 1MW or 1GW
  - say 10 GFLOPS/Watt on a <GPGPU,problem>
  - now take 1000 GPUGPUs to the same problem
  - realized perf is < 1000x (less than perfect parallelism)
  - required power > 1000x (energy to move data & heat)
- Scaling down not always easier with real constraints

if problem changes all bets off

Pay attention to denominator of normalized metrics

Comparing and Summarizing Performance

Relative Performance

- Performance = 1 / Time
  - shorter latency $\Rightarrow$ higher performance
  - higher throughput (job/time) $\Rightarrow$ 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

Architect's Definition of Faster

- "X is n times faster than Y" means

  n = Performance$_X$ / Performance$_Y$

  = Throughput$_X$ / Throughput$_Y$ *if rate*

  = Time$_Y$ / Time$_X$ *if latency*

- "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"

According to H&P

Architect's Definition of Speedup

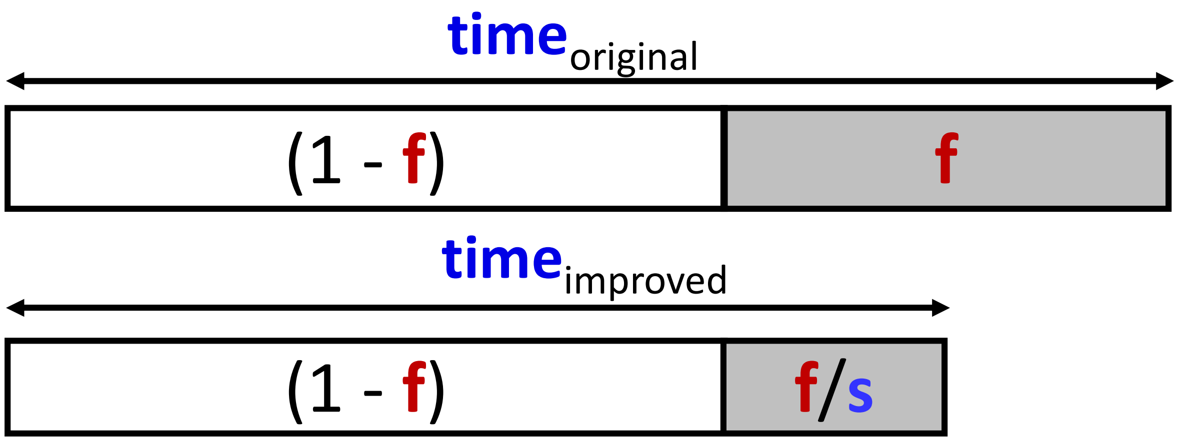
- If X is an "enhanced" version of Y, the "speedup" due to the enhancement is

$$S = \text{Time}_{\text{without enhancement}} / \text{Time}_{\text{with enhancement}}$$

$$= \text{Time}_Y / \text{Time}_X$$

Amdahl's Law: a lesson on speedup

- If only a fraction **f** (of time) is speedup by **s**



$$\text{time}_{\text{improved}} = \text{time}_{\text{original}} \cdot ((1-f) + f/s)$$

$$S_{\text{effective}} = 1 / ((1-f) + f/s)$$

  - if **f** is small, **s** doesn't matter
  - even when **f** is large, diminishing return on **s**; eventually "1-**f**" dominates

Amdahl's Law: a quiz

True or False:

An opcode X is used <u>infrequently</u> (less than 1 in 500 executed instructions) in an embedded workload. Amdahl's Law would say NOT to worry about optimizing the executions of opcode X on a processor designed specifically for that workload.

Hint: what does **f** mean in Amdahl's Law?

Summarizing Performance

- When comparing two computers X and Y, the relative performance of X and Y depends on program executed
  - X can be m% faster than Y on prog A
  - X can be n% (where m!=n) faster than Y on prog B
  - Y can be k% faster than X on prog C
- Which computer is faster and by how much?
  - depends on which program(s) you care about
  - if multiple programs, also depends their relative importance (frequency or occupancy??)
- Many ways to summarize performance comparisons into a single numerical measure
  - know what the resulting "number" actually mean
  - know when to use which to be meaningful

Arithmetic Mean

- Suppose workload is applications $A_0, A_1, \ldots, A_{n-1}$
- Arithmetic mean of run time is

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

  – comparing AM same as comparing total run-time

  caveat: longer running apps have greater contribution than shorter running apps

- If $AM_X/AM_Y=n$ then Y is n times faster than X . . .

  True: $A_0, A_1, \ldots, A_{n-1}$ run equal number of times always

  False: some apps run more frequently

  Especially bad if most frequent apps also shortest

Weighted Arithmetic Mean

- Describe relative frequency of apps by weights $w_0, w_1, \ldots, w_{n-1}$
  - w_i = number of A_i executions / total app executions
  - $\sum_{i=0}^{n-1} w_i = 1$
- Weighted AM of the run time = $\sum_{i=0}^{n-1} w_i \cdot Time_{A_i}$
- 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 always known, so why not "normalize"

$$\frac{1}{n}\sum_{i=0}^{n-1}\frac{Time_{A_i\ on\ X}}{Time_{A_i\ on\ Y}} \quad \text{or} \quad \sqrt[n]{\prod_{i=0}^{n-1}\frac{Time_{A_i\ on\ X}}{Time_{A_i\ on\ Y}}}$$

What does it mean though?

Danger of Normalized Performance

- Suppose
  - A_0 takes 1s on X; 10s on Y; 20s on Z
  - A_1 takes 1000s on X; 100s on Y; 20s 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 blindly take AM of rates or normalize metrics
  - 30mph drive to school (10 miles) and 90mph to return home, roundtrip average speed is not

    (30mph + 90mph)/2

same distance different time each way

- To compute average mph, expand fully

  average speed = total distance / total time

  = 20 / (10/30 + 10/90) = 45mph

- In case you are not confused,
  - if A_1@IPC_1, A_2@IPC_2, ...
  - what is $IPC_{average}$ if A_1, A_2, ... are equal in **# cyc** vs **# inst** vs **# occurrence**

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

What is IPC_{avg} of A_1@IPC_1 ... A_N@IPC_N ?

- If **k cycles** each:

$$\#inst_{total}/\#cyc_{total} = (\mathbf{k}\cdot IPC_1 + \ldots + \mathbf{k}\cdot IPC_N) / (\mathbf{k}\cdot\mathbf{N})$$
$$= (IPC_1 + \ldots\ IPC_N) / \mathbf{N}$$

- If **k instructions** each:

$$\#inst_{total}/\#cyc_{total} = \mathbf{k}\cdot\mathbf{N} / (\mathbf{k}/IPC_1 + \ldots + \mathbf{k}/IPC_N)$$
$$= \mathbf{N} / (1/IPC_1 + \ldots + 1/IPC_N)$$

- If **k occurrences** each: don't know without #inst or #cyc of a program occurrence

Forget equations, think what you want to know

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
- E.g. SPEC Benchmarks (www.spec.org)
  - 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

SPEC CPU Benchmark Suites

- CINT2006 (C or C++)

  perlbench (prog lang), bzip2 (compress), gcc (compile),mcf (optimize), gobmk (go), hmmer (gene seq. search), sjeng (chess), libquantum (physics sim.), h264ref (video compress), omnetpp (discrete event sim.), astar (path-finding), xalancbmk (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 GM of performance normalized to a 1997-era 296MHz Sun UltraSparc II

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

Performance Recap

- 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
  - explain what effect you are trying to measure
  - explain what and how you actually measured
  - explain how performance is summarized and represented

When it matters, people will check for themselves

Most important is to be truthful

We, the members of the IEEE, in recognition of the importance of our technologies . . . do hereby commit ourselves to the highest ethical and professional conduct and agree:

> *7. to be honest and realistic in stating claims or estimates based on available data;*

--- Paragraph 7.8 IEEE Code of Ethics, IEEE Policies

Bad to fool others; even worse to fool yourself