Lecture 4:
Performance of computer:
How to measure and compare?

Jangwoo Kim (Seoul National University)
jangwoo@snu.ac.kr

Announcement

- **I will be on business trip next week**
  - No lectures for 3/27, 3/29 ☺
  - The lab will be held on 3/27 as scheduled

- **We have a make-up lecture tonight (3/20)**
  - 3/27 (Thursday 9:30AM) lecture
    → 3/20 (Tuesday 5PM)
  - 3/29 (Thursday 9:30AM) lecture
    → We will find another slot soon

Sorry for this sudden notice. :<

Announcement

- HW #1
  - Posted on eTL
    - Due: 3/27 (in the beginning of the lecture)
      (No late homework will be accepted)
  - We will grade <u>your efforts</u>
    - Tell us your reasoning!
    - Simple mistakes are OK, but cheating is NOT.

- Skip Chapter 3!
  - To be covered later
  - We will cover more architecture-oriented stuff first.

How to measure performance of computer?

"How fast is your computer?"

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. race car)
  - Improving latency does not necessarily improve throughput
  - Improving throughput does not necessarily improve latency
- Not completely distinct though
  - Increasing throughput of component processing shortens the latency of the overall task

It is all about time

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

- UNIX "time" command
  - Elapsed time // wall-clock time
  - User CPU time // time spent running <u>your code</u>
  - System CPU time // time spent running other code <u>on behalf of your code</u>
  - 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. Measure & report wall-clock time on unloaded system

Example: Unix 'time' command

- @ Unix

  > time yes "hello"

```
hello
hello
hello
…
hello
hello
…
…
Real   0m1.260s
User   0m0.002s
Sys    0m0.013s
```

IPC, MIPS and GHz

- The metrics you are most likely to see
  - IPC (instruction per cycle) or CPI (cycle per instruction)
  - MIPS (million instruction per second)
  - GHz (10^9 cycles per second)

- **Iron Law** on Performance

(ISA & compiler)

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

1/GHz *1/MIPS* *CPI (= 1/IPC)*

(semiconductor tech & uArchitecture) *(uArchitecture & ISA)*

  - MIPS and IPC are averages
  - There factors are highly related as design trade-offs

IPC, MIPS and GHz

- The metrics you are most likely to see
  - IPC (instruction per cycle) or CPI (c
  - MIPS (million instruction per
  - GHz (10^9 cycles per s

- **Iron Law**

This is for 'single-core latency-performance' only!

(ISA & compiler)

(time/cyc) x (cyc/inst) x (inst/program)

1/GHz
(semiconductor tech & uArchitecture)

1/MIPS

CPI (= 1/IPC)
(uArchitecture & ISA)

  - MIPS and IPC are averages
  - There factors are highly related as design trade-offs

Case of FLOPS

- Scientific-computing people often use FLOPS

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

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

- Is this a good and fair metric to compare performance?
  - Yes, only if we are talking about the **same** problem.
    - Not all FFT algorithms have the same number of FP ops
    - Not all FP operations are equal
      (FADD vs. FMULT vs. FDIV)

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 usage
  - But never use more energy to run slower
- Some metrics of interest to consider power usage
  - Power = Energy / Time
  - Energy-delay-product = Energy * Time   // overall small?
  - In general, **function of** (Energy, Time)
- Other factors
  - Implementation costs, reliability, security, …

For this lecture, we care performance only.

How to summarize and compare performance?

"How much faster is computer A than B?"
"One performance number for many programs?"

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

  $n = \mathbf{Performance_X} / Performance_Y$

  $= \mathbf{Throughput_X} / Throughput_Y$

  or $= Time_Y / \mathbf{Time_X}$

- "X is m% faster than Y" means

  $1+m/100 = Performance_X / Performance_Y$

- To avoid confusion, focus on this definition of "**faster**"
  - Performance of Y for case 1

    $(Time_X=1) / (Time_Y= 0.5) = 2$ → "Y is 100% faster than X"
  - Performance of Y for case 2

    $(Time_X=1) / (Time_Y= 0.66) = 1.5$ → "Y is 50% faster than X"

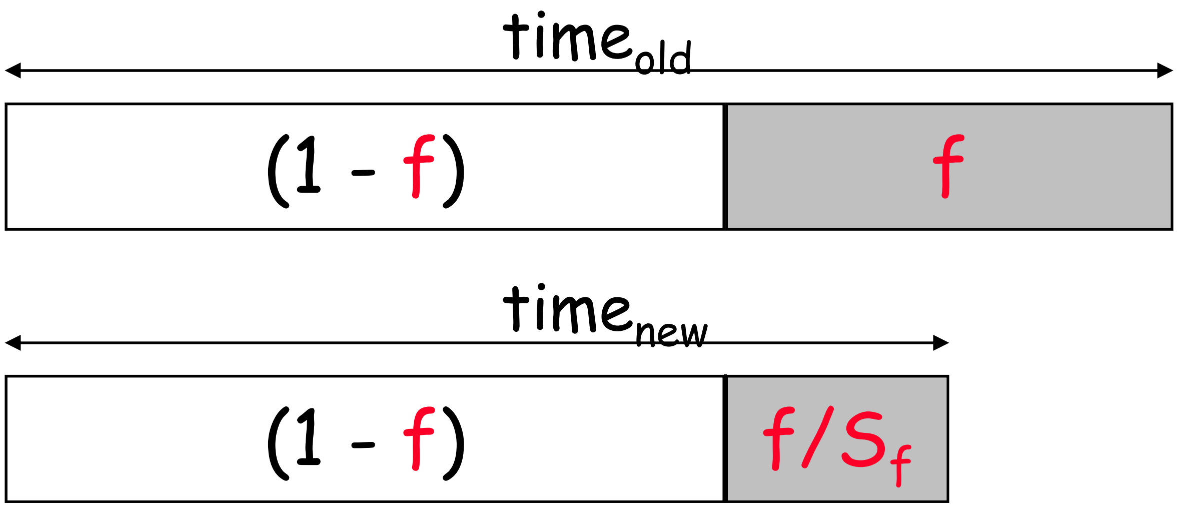
Speedup

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

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

Amdahl's Law on Speedup

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



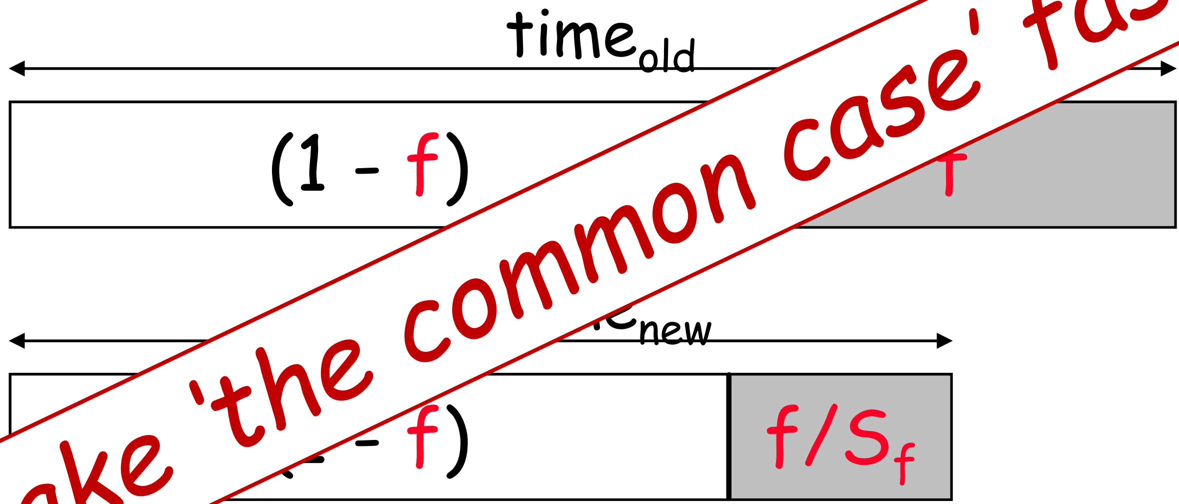
$$time_{new} = time_{old} \cdot ((1-f) + f/S_f)$$

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

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

Amdahl's Law on Speedup

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



Make 'the common case' faster!

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

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

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 there are several applications, relative importance matters.
- 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 your workload is applications $\mathbf{A_0, A_1, .. A_{n-1}}$
- 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
  - Issue: longer applications have greater contribution than shorter applications

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

  True: $A_0, \ldots A_{n-1}$ 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, $\mathbf{w_0,w_1 \ldots w_{n-1}}$ where $1=\sum_{i=0}^{n-1} w_i$
- $\mathbf{w_i}$ is the number of times $\mathbf{A_i}$ 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 $\mathbf{w_0,w_1 \ldots w_{n-1}}$

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

Geometric Mean

- Suppose your workload is applications $\mathbf{A_0, A_1, .. A_{n-1}}$
- Geometric Mean (GM) of the run time is

$$\sqrt[n]{\prod_{i=0}^{n-1} \text{Normalized execution time on } i} \rightarrow \sqrt[n]{\prod_{i=0}^{n-1} \frac{Time_{A_i \ on \ X}}{Time_{A_i \ on \ Y}}}$$

(1) Normalized the performance of X to Y for each workload

(2) Report the performance of computer X
as 'relative speedup' over the reference computer Y.

Note: GM (X_i) / GM (Y_i) = GM (X_i / Y_i)

Geometric Mean (continued)

- 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	0.63	1	1	0.63	1.58	1.58	1

Q1. Performance of X == Performance of Y ?

Q2. Treat (1000s → 500s) == (1s → 0.5s) ?

Tricky…..

Harmonic Mean

- Don't take arithmetic mean of "rates" (e.g. throughput)
  - e.g. 30 km/h for first 10 kms, 90 km/h for next 10 kms,
    the average speed is not (30 + 90)/2 = 60 km/h!

- To compute average rate
  1. Expand fully
     average speed = total distance / total time
     = 20 / (10/30 + 10/90) = 45 km/h
  2. Harmonic mean & weighed harmonic mean

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

Hmm.. Do these HM equations match the example?

Why HM for IPC?

- For **same frequency** and **same # of instructions**,

  Avg. Time = Avg. CPI = A.M.(CPI)

  - "Average" IPC = $\dfrac{1}{\text{A.M.(CPI)}}$

$$= \frac{1}{\frac{CPI_1}{n} + \frac{CPI_2}{n} + \frac{CPI_3}{n} + \ldots + \frac{CPI_n}{n}}$$

$$= \frac{n}{CPI_1 + CPI_2 + CPI_3 + \ldots + CPI_n}$$

$$= \frac{n}{\frac{1}{IPC_1} + \frac{1}{IPC_2} + \frac{1}{IPC_3} + \ldots + \frac{1}{IPC_n}} = \text{H.M.(IPC)}$$

Key summary

- Arithmetic Mean for “Amount”
  - Latency, time
- Geometric Mean for “Ratio”
  - Based on normalization
- Harmonic Mean for “Rate”
  - Speed, IPC

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

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 **SPECratio**: <u>geometric mean</u> of performance normalized to a 296MHz reference Sun UltraSparc II

Performance Summary

- There is no best-for-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 your results are interesting and convincing,
people will check it for themselves

Question?

Announcements: Homework #1 to be collected on Mar 27 (Tues).

Reading: start reading P&H Ch.4

Handouts: None