Lecture 2
Performance

Fall 2016

Pejman Lotfi-Kamran

Adapted from slides originally developed by Profs. Falsafi, Hill, Hoe, Lipasti, Shen, Smith, Sohi, and Vijaykumar of Carnegie Mellon University, EPFL, Purdue University, and University of Wisconsin.

Where Are We?

Fr	Sa	Su	Mo	Tu
	27-Shahrivar		29-Shahrivar	
	3-Mehr		5-Mehr	
	10-Mehr		12-Mehr	
	17-Mehr		19-Mehr	
	24-Mehr		26-Mehr	
	1-Aban		3-Aban	
	8-Aban		10-Aban	
	15-Aban		17-Aban	
	22-Aban		24-Aban	
	29-Aban		1-Azar	
	6-Azar		8-Azar	
	13-Azar		15-Azar	
	20-Azar		22-Azar	
	27-Azar		29-Azar	
	4-Dey		6-Dey	

This Lecture
- Performance measurement

Next Lecture:
- Basic caches

Metrics of Performance

Time (latency)
- elapsed time vs. processor time

Rate (bandwidth or throughput)
- performance = rate = work per time

Distinction is sometimes blurred
- consider batched vs. interactive processing
- consider overlapped vs. non-overlapped processing
- may require conflicting optimizations

What is the most popular metric of performance?

The "Iron Law" of Processor Performance

Processor Performance = Time / Program

$$= \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Time}}{\text{Cycle}}$$

(code size) **(CPI)** **(cycle time)**

Architecture --> Implementation --> Realizatior

Compiler Designer **Processor Designer** **Chip Designer**

MIPS

MIPS - Millions of Instructions per Second

$$\text{MIPS} = \frac{\frac{\text{\# of instructions}}{\text{benchmark}} \times \frac{\text{benchmark}}{\text{total run time}}}{1{,}000{,}000}$$

When comparing two machines (A, B) with the same instruction set, MIPS is a fair comparison (*sometimes…*)

But, MIPS can be a "*meaningless indicator of performance…*"

- instruction sets are not equivalent
- different programs use a different instruction mix
- instruction count is not a reliable indicator of work
  - some optimizations add instructions
  - instructions have varying work

MIPS (Cont.)

Example:

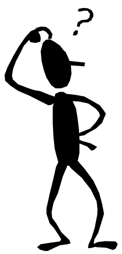
- Machine A has a special instruction for performing square root
  - It takes 100 cycles to execute
- Machine B doesn't have the special instruction
  - must perform square root in software using simple instructions
  - e.g, Add, Mult, Shift each take 1 cycle to execute

- Machine A: 1/100 MIPS = 0.01 MIPS
- Machine B: 1 MIPS

How about Relative MIPS = $(\text{time}_{\text{reference}}/\text{time}_{\text{new}}) \times \text{MIPS}_{\text{reference}}$?

MFLOPS

MFLOPS = (FP ops/program) x (program/time) x 10^{-6}



Popular in scientific computing

- There was a time when FP ops were much much slower than regular instructions (i.e., off-chip, sequential execution)

Not great for "predicting" performance because it

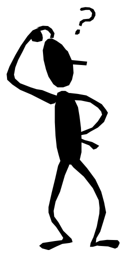
- ignores other instructions (e.g., load/store)
- not all FP ops are created equally
- depends on how FP-intensive program is

Beware of "peak" MFLOPS!

Normalized MFLOPS

Normalized FP: give canonical # FP ops to prog

Normalized MFLOPS = (# canonical FP ops/time) x 10^{-6}



Not all machines have the same FP ops

- Cray does not implement divide
- Motorola has SQRT, SIN, and COS

Not all FP ops do the same amount of work

- adds usually faster than divide

Used to compare performance, essentially a measure of execution time.

Comparing Performance

Often, we want to compare the performance of different machines or different programs. Why?

- help architects understand which is "better"
- give marketing a "silver bullet" for the press release
- help customers understand why they should buy <my machine>

Comparing Performance (Cont.)

If machine A is 50% slower than B, and $time_A$=1.0s, does that mean

case 1: $time_B$= 0.5s since $time_B/time_A$=0.5

case 2: $time_B$ = 0.666s since $time_A/time_B$=1.5

What if I say machine B is 50% faster than A?

If machine A is 1000% faster than B, and $time_A$=1.0s, does than mean

case 1: $time_B$=10.0s A is 10 times faster

case 2: $time_B$=11.0s A is 11 times faster

By Definition

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

Remember to compare "performance"

Performance vs. Execution Time

Often, we use the phrase "*X is faster than Y*"

- Means the response time or execution time is lower on X than it is on Y
- Mathematically, "X is N times faster than Y" means

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

$$\frac{\text{Execution Time}_Y}{\text{Execution Time}_X} = N = \frac{1/\text{Performance}_Y}{1/\text{Performance}_X} = \frac{\text{Performance}_X}{\text{Performance}_Y}$$

Performance and Execution time are *reciprocals*

- Increasing performance decreases execution time

Reasons to Compare Performance

To make a decision

- customers: which machine to buy
- designers: which optimization to include vs. leave out

Important to have "representative" performance

- Time/prog = insts/prog x cycles/inst x sec/cycle

Type of Benchmarks

Mixes

- instruction frequency of occurrence; calculate

Kernels

- "representative" program fragments
- good for focusing on individual features not big picture

Synthetic benchmarks

- programs intended to give specific mix
- ignore dependences
- maybe ok for non-pipelined, non-cached, w/o optimizing compilers
- questionable validity
- simple to measure and report

Type of Benchmarks (cont.)

Real programs

- representative of real workload
- more accurate way to characterize performance
- requires considerable work

But, nothing beats testing out your bread&butter application!!

Mix Example

Gibson Mix, developed in 1950's at IBM

load/store	31%	branch	17%
fixed add/sub	6%	compare	4%
float add/sub	7%	float mult	4%
float div	2%	fixed mult	1%
fixed div	< 1%	shifts	4%
logical	2%		

Generally speaking, these numbers are still valid today.

But, machine behaviors are too complicated

Kernel Example

Inner product

```
DO L = 1, LP
   Q = 0.0
DO K = 1, N
   Q = Q + Z(K)*X(K)
```

Lecture 2
Slide 17

Synthetic Benchmark Example

Dhrystone, Whetstone

```
X = 1.0
Y = 1.0
Z = 1.0
DO I = 1, N8
   CALL P3(X,Y,Z)
```

```
SUBROUTINE P3(X,Y,Z)
X1 = X
Y1 = Y
X1 = T * (X1 - Y1)
Y1 = T * (X1 + Y1)
Z = (X1 + Y1)/T2
RETURN
```

Intended to measure procedure call and return

How about X = SQRT(EXP (ALOG(X)/T1)) ?

Lecture 2
Slide 18

SPEC

SPEC - The System Performance Evaluation Cooperative (SPEC)

- founded in 1988 by a small number of workstation vendors who realized that the marketplace was in desperate need of realistic, standardize performance tests.
- Grown to become successful performance standardization bodies with more than 40 member companies. **http://www.spec.org**

SPEC's Philosophy

- The goal of SPEC is to ensure that the marketplace has a fair and useful set of metrics to differentiate candidate systems.
- The basic SPEC methodology is to provide the benchmarker with a standardized suite of source code based upon existing applications

Benchmarks come in many different flavors

- CPUINT && CPUFP 2006
- JVM08
- SFS – NFS benchmarks
- WEB09

Lecture 2
Slide 19

SPEC CPUInt 2006 Benchmarks

400.perlbench	C	Programming language
401.bzip2	C	Compression
403.gcc	C	C Programming Language Compiler
429.mcf	C	Combinatorial Optimization
445.gobmk	C	Artificial Intelligence: Go
456.hmmer	C	Search Gene Sequence
458.sjeng	C	Artificial Intelligence: Chess
462.libquantum	C	Physics / Quantum Computing
464.h264ref	C	Video Compression
471.omnetapp	C++	Discrete Event Simulation
473.astar	C++	Path-Finding Algorithms
483.xalancbmk	C++	XML Processing

Lecture 2
Slide 20

SPEC CPUFP 2006 Benchmarks

410.bwaves	Fortran	Fluid Dynamics
416.gamess	Fortran	Quantum Chemistry
433.milc	C	Physics / Quantum Chromodynamics
434.zeusmp	Fortran	Physics / CFD
435.gromacs	C, Fortran	Bio Chemistry / Molecular Dynamics
436.cactusADM	C, Fortran	Physics / General Relativity
437.leslie3d	Fortran	Fluid Dynamics
444.Namd	C++	Biology / Molecular Dynamics
447.dealII	C++	Finite Element Analysis
450.soplex	C++	Linear Programming, Optimization
453.povray	C++	Image Ray-Tracing
454.calculix	C, Fortran	Structural Mechanics
459.GemsFDTD	Fortran	Computational Electromagnetics
465.tonto	Fortran	Quantum Chemistry
470.lbm	C	Fluid Dynamics
481.wrt	C, Fortran	Weather
482.sphinx3	C	Speech Recognition

SPEC Performance Numbers

Geometric Mean of 12 (SpecINT) and 17 (SpecFP) Benchmarks

- Performance measured against Sun UltraSparc II system at 296MHz

2006 Performance Numbers (www.spec.org)

	AMD Optron 2.4GHz	IBM Power 5 1.9GHz	Intel Itanium 2 1.5GHz	Intel Pentium 4 3.4GHz	SGI R14K 500MHz	Sun Ultra III Cu 1.2GHz
Int*	1346	1398	1408	1669	483	642
FP*	1428	2576	2038	1544	362	877

*** results are for SPEC 2000 benchmarks**

CloudSuite 1.0

(released @ parsa.epfl.ch/cloudsuite)

Data Serving
Cassandra NoSQL

MapReduce
Machine learning on Hadoop

Media Streaming
Apple Quicktime Server

SAT Solver
Symbolic VM constraint

Web Frontend
Nginx, PHP server

Web Search
Apache Nutch

Comparing Multiple Programs

	Computer A	Computer B	Computer C
Program 1 (secs)	1	10	20
Program 2 (secs)	1000	100	20
Program 3 (secs)	1001	110	40

A is 10 times faster than B for program 1
B is 10 times faster than A for program 2
A is 20 times faster than C for program 1
C is 50 times faster than A for program 2
B is 2 times faster than C for program 1
C is 5 times faster than B for program 2

Each statement above is correct…,

…but I just want to know which machine is the best?

Let's Try a Simpler Example

Two machines timed on two benchmarks

	Machine A	Machine B
Program 1	2 seconds	4 seconds
Program 2	12 seconds	8 seconds

- How much faster is Machine A than Machine B?
- Attempt 1: **ratio of run times, normalized to Machine A times**

program1: 4/2 **program2 :** 8/12

- Machine A ran 2 times faster on program 1, 2/3 times faster on program 2
- On ***average***, Machine A is **(2 + 2/3) /2 = 4/3** times faster than Machine B

It turns this "averaging" stuff can fool us; watch…

Example (con't)

Two machines timed on two benchmarks

	Machine A	Machine B
Program 1	2 seconds	4 seconds
Program 2	12 seconds	8 seconds

- How much faster is Machine A than B?
- Attempt 2: **ratio of run times, normalized to Machine B times**

program 1: 2/4 **program 2 :** 12/8

- Machine A ran program 1 in 1/2 the time and program 2 in 3/2 the time
- On **average, (1/2 + 3/2) / 2 = 1**
- Put another way, Machine A is **1.0 times faster** than Machine B

Example (con't)

Two machines timed on two benchmarks

	Machine A	Machine B
Program 1	2 seconds	4 seconds
Program 2	12 seconds	8 seconds

- How much faster is Machine A than B?
- Attempt 3: **ratio of run times, aggregate (total sum) times, norm. to A**
  - Machine A took 14 seconds for both programs
  - Machine B took 12 seconds for both programs
  - Therefore, Machine A takes 14/12 of the time of Machine B
  - Put another way, Machine A is **6/7 faster** than Machine B

Which is Right?

Question:

- How can we get **three different answers**?

Solution

- Because, while they are all **reasonable** calculations…

…each answers a *different* question

We need to be more precise in understanding and posing these performance & metric questions

Arithmetic and Harmonic Mean

Average of the execution time that tracks total execution time is the arithmetic mean

$$\frac{1}{n}\sum_{i=1}^{n} Time_i$$

This is the defn for "average" you are most familiar with

If performance is expressed as a rate, then the average that tracks total execution time is the harmonic mean

$$\frac{n}{\sum_{i=1}^{n}\frac{1}{Rate_i}}$$

This is a different defn for "average" you are prob. less familiar with

Lecture 2
Slide 29

Problems with Arithmetic Mean

Applications do not have the same probability of being run

Longer programs weigh more heavily in the average

For example, two machines timed on two benchmarks

	Machine A	Machine B
Program 1	2 seconds (20%)	4 seconds (20%)
Program 2	12 seconds (80%)	8 seconds (80%)

- If we do arithmetic mean, Program 2 "counts more" than Program 1
  - an improvement in Program 2 changes the average more than a proportional improvement in Program 1
- But perhaps Program 2 is 4 times more likely to run than Program 1

Lecture 2
Slide 30

Weighted Execution Time

Often, one runs some programs more often than others. Therefore, we should weight the more frequently used programs' execution time

$$\sum_{i=1}^{n} Weight_i \times Time_i$$

Weighted Harmonic Mean

$$\frac{1}{\sum_{i=1}^{n}\frac{Weight_i}{Rate_i}}$$

Lecture 2
Slide 31

Using a Weighted Sum (or weighted average)

	Machine A	Machine B
Program 1	2 seconds (20%)	4 seconds (20%)
Program 2	12 seconds (80%)	8 seconds (80%)
Total	10 seconds	7.2 seconds

- Allows us to determine relative performance 10/7.2 = 1.38
  --> Machine B is 1.38 times faster than Machine A

Lecture 2
Slide 32

Quiz

E.g., 30 mph for first 10 miles, 90 mph for next 10 miles

What is the average speed?

Average speed = (30 + 90)/2 = 60 mph *(Wrong!)*

The correct answer:

Average speed = total distance / total time

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

= 45 mph

For rates use Harmonic Mean!

Another Solution

Normalize runtime of each program to a reference

	Machine A (ref)	Machine B
Program 1	2 seconds	4 seconds
Program 2	12 seconds	8 seconds
Total	10 seconds	7.2 seconds

	Machine A (norm to B)	Machine B (norm to A)
Program 1	0.5	2.0
Program 2	1.5	0.666
Average?	1.0	1.333

So when we normalize A to B, and average, it looks like A & B are the same.
But when we normalize B to A, it looks like A is better!

Geometric Mean

Used for relative rate or performance numbers

$$Relative\_Rate = \frac{Rate}{Rate_{ref}} = \frac{Time_{ref}}{Time}$$

Geometric mean

$$\sqrt[n]{\prod_{i=1}^{n} Relative\_Rate_i} = \frac{\sqrt[n]{\prod_{i=1}^{n} Rate_i}}{Rate_{ref}}$$

Using Geometric Mean

	Machine A (norm to B)	Machine B (norm to A)
Program 1	0.5	2.0
Program 2	1.5	0.666
Geometric Mean	0.866	1.155

1.155 = 1/0.8666!

Drawbacks:

- Does not predict runtime because it normalizes
- Each application now counts equally

Well…..

Geometric mean of ratios is not proportional to total time

- Arithmetic mean in example says machine B is 1.166 times faster
- Geometric mean says they machine B is 1.155 times faster

Rule of thumb: Use AM for times, HM for rates, GM for ratios

SPEC Uses Geometric Mean

Steps:

1. for each benchmark i, look up $T_{base,i}$
2. for each benchmark i, run target machine to get $T_{new,i}$
3. compute geometric mean: $\sqrt[n]{\prod_{1}^{n} T_{base,i}/T_{new,i}}$

Is SPEC a good predictor of performance?

Summary

Performance is important to measure

- For architects comparing different deep mechanisms
- For developers of software trying to optimize code, applications
- For users, trying to decide which machine to use, or to buy

Performance metric are subtle

- Easy to mess up the "machine A is XXX times faster than machine B" numerical performance comparison
- You need to know exactly what you are measuring: time, rate, throughput, CPI, cycles, etc
- You need to know how combining these to give aggregate numbers does different kinds of "distortions" to the individual numbers
- No metric is perfect, so lots of emphasis on standard benchmarks today