Quantifying Performance

EEC 170 Fall 2005
Chapter 4

"I can't improve it if I don't know how to measure it"

Courtesy of Prof. John D. Owens, ECE dept, UC-Davis.

Performance

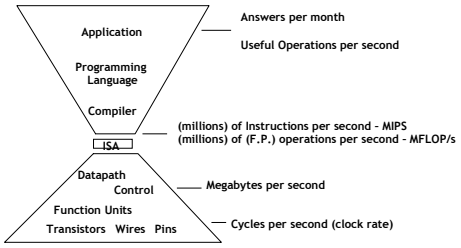
Measure, Report, and Summarize
Make intelligent choices
See through the marketing hype
Key to understanding underlying organizational motivation

Why is some hardware better than others for different programs?

What factors of system performance are hardware related? (e.g., Do we need a new machine, or a new operating system?)

How does the machine's instruction set affect performance?

Metrics of performance



Each metric has a place and a purpose, and each can be misused

Two notions of "performance"

Plane	DC to Paris	Speed	Passengers	Throughput (pmph)
Boeing 747	6.5 hours	610 mph	470	286,700
BAD/Sud Concorde	3 hours	1350 mph	132	178,200

Which has higher performance?

- Time to do the task (Execution Time)
  - execution time, response time, latency
- Tasks per day, hour, week, sec, ns … (Performance)
  - throughput, bandwidth
- Response time and throughput often are in opposition

Example

Time of Concorde vs. Boeing 747?

- Concorde is 1350 mph / 610 mph = 2.2 times faster
- = 6.5 hours / 3 hours

Throughput of Concorde vs. Boeing 747 ?

- Concorde is 178,200 pmph / 286,700 pmph = 0.62 "times faster"
- Boeing is 286,700 pmph / 178,200 pmph = 1.60 "times faster"

Boeing is 1.6 times ("60%") faster in terms of throughput
Concord is 2.2 times ("120%") faster in terms of flying time
We will focus primarily on execution time for a single job

- But sysadmins may use throughput as their primary metric!

Latency vs. Throughput

Latency (Response Time)

- How long does it take for my job to run?
- How long does it take to execute a job?
- How long must I wait for the database query?

Throughput

- How many jobs can the machine run at once?
- What is the average execution rate?
- How much work is getting done?

If we upgrade a machine with a new processor what do we increase?

If we add a new machine to the lab what do we increase?

Definitions

Performance is in units of things-per-time

- Miles per hour, bits per second, widgets per day …
- Bigger is better

If we are primarily concerned with response time:

- Performance(x) = 1 / ExecutionTime(x)

"X is n times faster than Y" means

- n = Performance(X) / Performance(Y) = Speedup
- If X is 1.yz times faster than Y, we can informally say that X is yz% faster than Y. Speedup is better.

Execution Time

Elapsed Time

- counts everything *(disk and memory accesses, I/O, etc.)*
- a useful number, but often not good for comparison purposes

CPU time

- doesn't count I/O or time spent running other programs
- can be broken up into *system time* and *user time*

```
% /usr/bin/time du -s
81329656        .
      104.44 real         0.50 user         9.86 sys
```

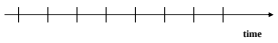
Our focus: user CPU time

- time spent executing the lines of code that are "in" our program

Clock Cycles

Instead of reporting execution time in seconds, we often use cycles:

$$\frac{\text{seconds}}{\text{program}} = \frac{\text{cycles}}{\text{program}} \times \frac{\text{seconds}}{\text{cycle}}$$



Clock "ticks" indicate when to start activities

Cycle time = time between ticks = seconds per cycle

Clock rate (frequency) = cycles per second (1 Hz = 1 cycle/sec)

- A 200 MHz clock has a cycle time of …

$$\frac{1}{200 \times 10^6} \times 10^9 = 5 \text{ nanoseconds}$$

Clock Speed Is Not The Whole Story

	SPECint95	*SPECfp95*
195 MHz MIPS R10000	11.0	17.0
400 MHz Alpha 21164	12.3	17.2
300 MHz UltraSPARC	12.1	15.5
300 MHz Pentium II	11.6	8.8
300 MHz PowerPC G3	14.8	11.4
135 MHz POWER2	6.2	17.6

[http://www.pattosoft.com.au/Articles/ModernMicroprocessors/]

How to Improve Performance

$$\frac{\text{seconds}}{\text{program}} = \frac{\text{cycles}}{\text{program}} \times \frac{\text{seconds}}{\text{cycle}}$$

So, to improve performance (everything else being equal) you can either (increase/ decrease):

_____ the # of required cycles for a program, or
_____ the clock cycle time or, said another way,
_____ the clock rate.

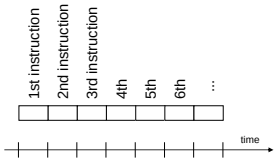
Clock Rate

Comparing processor performance for same or different architectures using clocks rate is invalid. Ignores instruction count and CPI, e.g.:

- 2.4GHz AMD Athlon processor is faster than 3.4GHz Intel Pentium 4 executing floating-point code (P4 has higher CPI)

How many cycles in a program?

Could assume that # of cycles = # of instructions



This assumption is incorrect:

- different instructions take different amounts of time on different machines (even with the same instruction set).
- Why?

Different #s of cycles for diff'nt instrs

Multiplication takes more time than addition

Floating point operations take longer than integer ones

Accessing memory takes more time than accessing registers

Important point: changing the cycle time often changes the number of cycles required for various instructions (more later)

Example instruction latencies

Imagine Stream Processor:

On ALU:

- **Integer adds: 2 cycles**
- **FP adds: 4 cycles**
- **Logic ops (and, or, xor): 1**
- **Equality: 1**
- **< or >: 2**
- **Shifts: 1**
- **Float->int: 3**
- **Int->float: 4**
- **Select (a?b:c): 1**

Other functional units:

- **Integer multiply: 4**
- **Integer divide: 22**
- **Integer remainder: 23**
- **FP multiply: 4**
- **FP divide: 17**
- **FP sqrt: 16**

CPI

How many clock cycles, *on average*, does it take for every instruction executed?

We call this CPI ("Cycles Per Instruction").

Its inverse (1/CPI) is IPC ("Instructions Per Cycle").

CISC machines: this number is

- **high(er)**

RISC machines: this number is

- **low(er)**

CPI: Average Cycles per Instruction

CPI = (CPU Time * Clock Rate) / Instruction Count
= Clock Cycles / Instruction Count

$$CPI = \sum_{i=1}^{n} CPI_i \times F_i \quad \text{where} \quad F_i = \frac{I_i}{\text{Instruction Count}}$$

On Imagine, integer adds are 2 cycles, FP adds are 4.
Consider an application that has 1/3 integer adds and 2/3 FP adds.
What is its CPI?

Given a 3 GHz machine, how many instrs/sec?

Instruction Selection

Compiler typically can choose among various instruction sequences to maximize performance

Instruction Class	CPI for this Class
A	1
B	2
C	3

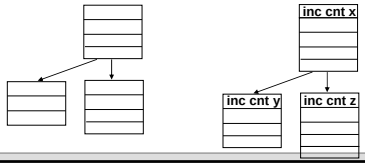
	Instruction Count in Class		
Code Sequence	A	B	C
1	2	1	2
2	4	1	1

- **Sequence 2 is longer, 6 vs. 5, but faster, 9 vs. 10**

Program Profiling

Can measure instruction count using software profiling tool:

- Compiler divides machine code into *Basic Blocks*, machine instruction sequence always executed together
- Inserts instructions that count each time a basic block is executed
- Instruction count per basic block = Count x Block size
- Total instruction count = Σ Basic block instruction counts



Millions of Instructions Per Second (MIPS)

MIPS = Clock Rate / CPI x 10^6

Ignores program instruction count. Hence, valid only for comparing processors running same object code.

Can vary inversely with performance!

- E.g., optimizing compiler eliminates instructions with relatively low cycle count. Causes CPI to increase and MIPS to decrease, but performance increases.

Sometime called Meaningless Indicator-of-Performance Statistic

megaFLOPS

Floating point performance metric: Million Floating Point Operations Per Second

- for given program rate of fadd, fmult, etc.

Not valid for different architectures because available floating point operations may differ

- E.g., some include div, sine, sqrt; other synthesize these operations with other simpler floating point operations
- E.g., some include a single multiply/accumulate

Peak megaFLOPS: rate guaranteed not to exceed, supercomputers may achieve only a few percent of peak. Very misleading.

The Performance Equation

Time = Clock Speed * CPI * Instruction Count

- = seconds/cycle * cycles/instr * instrs/program
- => seconds/program

"The only reliable measure of computer performance is time."

Now that we understand cycles …

A given program will require

- some number of instructions (machine instructions)
- some number of cycles
- some number of seconds

We have a vocabulary that relates these quantities:

- cycle time (seconds per cycle)
- clock rate (cycles per second)
- CPI (cycles per instruction)
  a floating point intensive application might have a higher CPI
- MIPS (millions of instructions per second)
  this would be higher for a program using simple instructions

Performance

Performance is determined by execution time

Do any of the other variables equal performance?

- # of cycles to execute program?
- # of instructions in program?
- # of cycles per second?
- average # of cycles per instruction?
- average # of instructions per second?

Common pitfall: thinking one of the variables is indicative of performance when it really isn't.

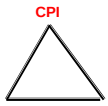
Evaluating Instruction Sets

Design-time Metrics:
- Can it be implemented, in how long, at what cost?
- Can it be programmed? Ease of compilation?

Static Metrics:
- How many bytes does the program occupy in memory?

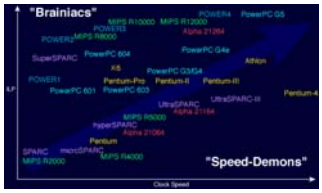
Dynamic Metrics:
- How many instructions are executed?
- How many bytes does the processor fetch to execute the program?
- How many clocks are required per instruction?
- How "lean" a clock is practical?

Best Metric: Time to execute the program!



NOTE: this depends on instruction set, processor organization, and compilation techniques.

Brainiacs vs. Speed Demons



[http://www.pattosoft.com.au/Articles/ModernMicroprocessors/]

Modern processor design balances CPI and clock speed
- Brainiacs do more work per clock cycle
- Speed demons have faster clock cycles

Aspects of CPU Performance

$$\text{CPU time} = \frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}$$

	Instr count	CPI	Clock rate
Program			
Compiler			
Instruction Set			
Organization			
Technology			

Aspects of CPU Performance

$$\text{CPU time} = \frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}$$

	Instr count	CPI	Clock rate
Program	X		
Compiler	X	X	
Instruction Set	X	X	X
Organization		X	X
Technology			X

Remember

Performance is specific to a particular program/s
- Total execution time is a consistent summary of performance

For a given architecture performance increases come from:
- increases in clock rate (without adverse CPI affects)
- improvements in processor organization that lower CPI
- compiler enhancements that lower CPI and/or instruction count

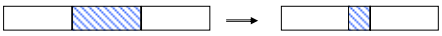
Pitfall:

expecting the improvement of one aspect of a computer to increase performance by an amount proportional to the size of improvement.

Amdahl's Law

Speedup due to enhancement E:

$$\text{Speedup(E)} = \frac{\text{ExTime w/o E}}{\text{ExTime w/ E}} = \frac{\text{Performance w/ E}}{\text{Performance w/o E}}$$



Suppose that enhancement E accelerates a fraction F of the task by a factor S and the remainder of the task is unaffected:

$$\text{ExTime (with E)} = ((1-F) + F/S) * \text{ExTime(without E)}$$

$$\text{Speedup (with E)} = \frac{1}{(1-F) + F/S}$$

Design Principle: Make the common case fast!

Undergrad Productivity

Average ECE student spends:

- 4 hours sleeping
- 2 hours eating
- 18 hours studying (yeah … right!)

Magic pill gives you all sleeping, eating in 1 minute!

How much more productive can you get?

Undergrad Productivity

$$\text{Speedup (with E)} = \frac{1}{(1-F) + F/S}$$

F = accelerated fraction = 0.25 (6 hrs/24 hrs)

S = speedup = 6 hrs / 1 minute = 360

Overall speedup:

- 1 / [(1-0.25) + (0.25/360)]
- ~= 1 / (1-0.25)
- ~= 1.33
- 33% more productive!

Benchmarks

Widely used programs for assessing execution time

Results must be reproducible => input is specified along with program

Benchmark types:

- Application programs: Most accurate example, SPEC benchmarks suite.
- Kernels: Key inner loops extracted from applications (Program typically spend 90% of time in 10% of code). Examples, Linpack & Livermore loops for supercomputers
- Toy benchmarks: Small enough to be entered by hand (e.g., 100 lines).
- Synthetic Benchmarks: Non-real programs based on instruction mix statistics

Problems with Small Benchmarks

Does not fully exercise memory hierarchy (e.g., entire program data set may fit in cache)

Encourages "benchmark specific optimizations" (cheating?) in compiler or processor hardware that do not benefit real applications => do not accurately predict application performance.

Proprietary Benchmarks

Results cannot be reproduced, leads to unproductive debate

"Benchmark wars erupt again"

By Frank Williams, Chicago Tribune

Tuesday, October 6, 1998

The benchmark wars in the world of computer hyping have erupted again. The latest major offensive is by PC Magazine. In reality this is but a belated counterattack to the Apple offensive of last November. Wintel platform proponents claim to have "proof" that their machines are faster than Macintoshes. Apple supporters claim to have "proof" that the processor inside every new Mac is twice as fast as those in Wintels....

SPEC Benchmark Suite

Cooperative formed by major companies (HP, SUN, IBM) to develop standard application set for Unix systems

- Later also used for Windows NT/2000/XP

SPEC CPU2000 includes set of 11 integer codes and 14 floating point codes, reported as SPECint and SPECfp

- Int programs include FPGA place and route, FP programs include crash simulation

Execution length > 10^{10} instructions per code, code size 10k-100k words and more. Exercises entire system, including compiler.

- Compiler improvements have yielded 10s of percent improvements for some systems

Reporting Performance

Result should be reproducible implies must specify:

- processor model
- memory configuration (including cache, main & disk)
- compiler version
- OS version

Best is to report execution time for all applications

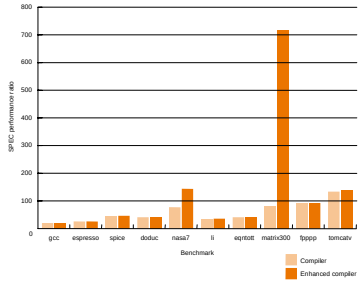
- Allows user of data to focus on data most relevant

Basis of Evaluation

Pros		Cons
• representative	**Actual Target Workload**	• very specific • non-portable • difficult to run, or measure • hard to identify
• portable • widely used • improvements useful in reality	**Full Application Benchmarks**	• less representative
• easy to run, early in design cycle	**Small "kernel" benchmarks**	• easy to "fool"
• identify peak capability and potential bottlenecks	**Microbenchmarks**	• "peak" may be a long way from application performance

SPEC '89

Compiler "enhancements" and performance



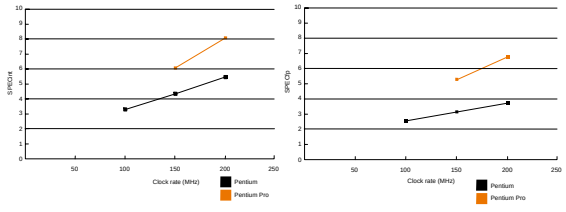
SPEC '95

Benchmark	Description
go	Artificial intelligence; plays the game of Go
m88ksim	Motorola 88k chip simulator; runs test program
gcc	The Gnu C compiler generating SPARC code
compress	Compresses and decompresses file in memory
li	Lisp interpreter
ijpeg	Graphic compression and decompression
perl	Manipulates strings and prime numbers in the special-purpose programming language Perl
vortex	A database program
tomcatv	A mesh generation program
swim	Shallow water model with 513 x 513 grid
su2cor	quantum physics; Monte Carlo simulation
hydro2d	Astrophysics; Hydrodynamic Naiver Stokes equations
mgrid	Multigrid solver in 3-D potential field
applu	Parabolic/elliptic partial differential equations
trub3d	Simulates isotropic, homogeneous turbulence in a cube
apsi	Solves problems regarding temperature, wind velocity, and distribution of pollutant
fpppp	Quantum chemistry
wave5	Plasma physics; electromagnetic particle simulation

SPEC '95

Does doubling the clock rate double the performance?

Can a machine with a slower clock rate have better performance?



SPEC CPU 2000 (Integer)

Benchmark	Description
gzip	Compression
vpr	FPGA circuit Place/Route
gcc	C compiler
mcf	Combinatorial optimization
crafty	Game playing: Chess
parser	Word processing
eon	Visualization (C++)
perlbmk	Perl
gap	Group theory interpreter
vortex	Object-oriented database
bzip2	Compression
twolf	Place/route simulator

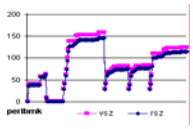
SPEC CPU 2000 (FP)

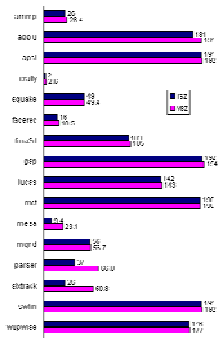
Benchmark	Description
wupwise	Physics/Quantum Chromodynamics (F77)
swim	Shallow water modeling (F77)
mgrid	Multi-grid solver: 3D potential field (F77)
applu	Parabolic/elliptic PDEs (F77)
mesa	3D graphics library (C)
galgel	Computational fluid dynamics (F90)
art	Image processing / neural networks (F90)
equake	Seismic wave propagation simulation (C)
facerec	Image processing: face recognition (F90)
ammp	Computational chemistry (C)
lucas	Number theory / primality test (F90)
fma3D	Finite-element crash simulation (F90)
sixtrack	High-energy nuclear physics accelerator design (F77)
apsi	Meteorology: pollutant distribution (F77)

SPEC 2000 Memory

Goals:

- Benchmarks larger than cache sizes
- No memory footprints > 200 MB





http://spec.unipv.it/cpu2000/analysis/memory/#graphs

Choosing SPEC Benchmarks

Good benchmarks …

- Have many users
- Exercise significant hardware resources
- Solve interesting technical problems
- Generate published results
- Add variety to benchmark suite

Bad benchmarks …

- Can't be ported in a reasonable time
- Are not compute bound (instead I/O bound)
- Have unchanged workloads from previous SPEC suites
- Are code fragments instead of complete apps
- Are redundant
- Do different work on different platforms

[http://spec.unipv.it/cpu2000/papers/COMPUTER_200007-abstract.JLH.html]

Calculating Overall SPEC Values

- **Normalize execution times to reference machine (SPEC '95: Sun SparcStation 10/40)**
- **Average normalized execution times**
- **But what do we mean by "average"?**

Calculating Overall SPEC Values

	Time on A	Time on B	A, norm to A	B, norm to A	A, norm to B	B, norm to B
Prog 1	1	10	1	10	0.1	1
Prog 2	1000	100	1	0.1	10	1
Arith mean	500.5	55	1	5.05	5.05	1

Calculating Overall SPEC Values

	Time on A	Time on B	A, norm to A	B, norm to A	A, norm to B	B, norm to B
Prog 1	1	10	1	10	0.1	1
Prog 2	1000	100	1	0.1	10	1
Arith mean	500.5	55	1	5.05	5.05	1
Geom mean	31.6	31.6	1	1	1	1

Calculating The Mean

Arithmetic mean (n items):

$[\ \Sigma\ (\text{Execution Time Ratio})_i\ ] / n$
$i = 1$ to n

Geometric mean (n items):

$[\ \Pi\ (\text{Execution Time Ratio})_i\ ]^{1/n}$
$i = 1$ to n

Averaging normalized execution times requires the geometric mean. Using it, mean of ratios and ratio of means gives the same result.

The geometric mean does *not* predict relative execution time.

Discussion Section

CPI Example

Suppose we have two implementations of the same instruction set architecture (ISA).

For some program,

- Machine A has a clock cycle time of 10 ns and a CPI of 2.0
- Machine B has a clock cycle time of 20 ns and a CPI of 1.2
- What machine is faster for this program, and by how much?

If two machines have the same ISA, for a given program which of our quantities (e.g., clock rate, CPI, execution time, # of instructions, MIPS) will always be identical?

of Instructions Example

A compiler designer is trying to decide between two code sequences for a particular machine.

- Based on the hardware implementation, there are three different classes of instructions: Class A, Class B, and Class C
- They require one, two, and three cycles (respectively).
- The first code sequence has 5 instructions: 2 of A, 1 of B, and 2 of C.
- The second sequence has 6 instructions: 4 of A, 1 of B, and 1 of C.

Which sequence will be faster? How much?
What is the CPI for each sequence?

MIPS example

Two different compilers are being tested for a 100 MHz machine with three different classes of instructions: Class A, Class B, and Class C, which require one, two, and three cycles (respectively). Both compilers are used to produce code for a large piece of software.

The first compiler's code uses 5 million Class A instructions, 1 million Class B instructions, and 1 million Class C instructions.

The second compiler's code uses 10 million Class A instructions, 1 million Class B instructions, and 1 million Class C instructions.

MIPS example

Which sequence will be faster according to MIPS?

Which sequence will be faster according to execution time?

Example (RISC processor)

Base Machine (Reg / Reg)

Op	Freq	Cycles	CPI(i)	% Time
ALU	50%	1		
Load	20%	5		
Store	10%	3		
Branch	20%	2		

Typical Mix

What's the CPI?

Example (RISC processor)

Base Machine (Reg / Reg)

Op	Freq	Cycles	CPI(i)	% Time
ALU	50%	1	.5	23%
Load	20%	5	1.0	45%
Store	10%	3	.3	14%
Branch	20%	2	.4	18%
			2.2	

Typical Mix

Example (RISC processor)

How much faster would the machine be if a better data cache reduced the average load time to 2 cycles?

Example (RISC processor)

How does this compare with using branch prediction to shave a cycle off the branch time?

Example (RISC processor)

What if two ALU instructions could be executed at once?

(different than deleting half the ALU ops!)

Example (Amdahl's Law 1)

Execution Time After Improvement =
Execution Time Unaffected + (Execution Time Affected / Amount of Improvement)

Example:
"Suppose a program runs in 100 seconds on a machine, with multiply responsible for 80 seconds of this time. How much do we have to improve the speed of multiplication if we want the program to run 4 times faster?"

How about making it 5 times faster?

Example (Amdahl's Law 2)

Suppose we enhance a machine making all floating-point instructions run five times faster. If the execution time of some benchmark before the floating-point enhancement is 10 seconds, what will the speedup be if half of the 10 seconds is spent executing floating-point instructions?

Example (Amdahl's Law 2)

We are looking for a benchmark to show off the new floating-point unit described in Part 2, and want the overall benchmark to show a speedup of 3. One benchmark we are considering runs for 100 seconds with the old floating-point hardware. How much of the execution time would floating-point instructions have to account for in this program in order to yield our desired speedup on this benchmark?

Example (Compiler Optimization)

You want to understand the performance of a specific program on your 3.3 GHz machine. You collect the following statistics for the instruction mix and breakdown:

Instruction Class	Frequency (%)	Cycles
Arithmetic/logical	50	1
Load	20	2
Store	10	2
Jump	10	1
Branch	10	3

Part A

Calculate the CPI and MIPS for this program.

Part B

Your compiler team reports they can eliminate 20% of ALU instructions (i.e. 10% of all instructions). What is the speedup?

Part C

With the compiler improvements, what is the new CPI and MIPS?