Measuring Performance

Tom Kelliher, CS 220

Sept. 4, 2009

1 Administrivia

Announcements

Assignment

No additional reading.

From Last Time

Introduction.

Outline

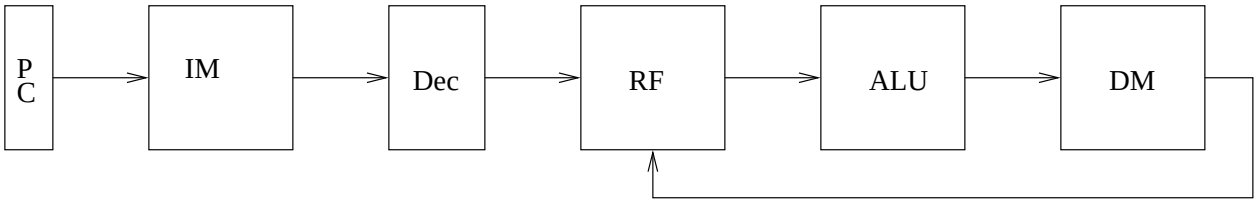
1. CPU model.
2. Defining performance.
3. Measuring performance.
4. Choosing benchmarks.

Coming Up

Comparing performance.

2 CPU Model

1. First off, multipliers: gig, meg, nano, pico.
2. What is the "clock"?
   (a) Clock frequency/rate. Clock period.
   (b) Logic gate circuit delays.
   Combinational and sequential logic.
   (c) How much work can I do in a clock period?
3. CPU model:



4. CPU implementations: single cycle, multiple cycle, pipelined.
   What is super-pipelining?
5. Relating this model to CPI.

3 Defining Performance

1. Why do we care about performance?

2. What is it? What do we measure?

   (a) GHz? Matters to marketing types.

   (b) How quickly we can run synthetic benchmarking kernels? (Toy programs.)

   (c) Throughput? Matters mostly to system admins.

   (d) Response time? Matters mostly to users.

3. Response time. Definition:

   Begin to finish time for a program, as measured by a "wall clock."

   Response time then includes:

   (a) I/O time.

   (b) Time CPU assigned to other users.

   (c) Time necessary for system tasks.

   Another measure of response time: user CPU time.

   System performance — elapsed time (wall time) on an unloaded system. Accounts for everything (I/O, users, OS overhead).

   CPU performance — user CPU time. Best metric for comparing processors?

4 Measuring Performance

Equations:

1. Performance:

$$\text{Performance} = \frac{1}{\text{Execution Time}}$$

   Higher numbers are better.

2. Relative performance (suppose machine A is faster than B):

$$\frac{\text{Performance}_\text{A}}{\text{Performance}_\text{B}} = \frac{\text{Execution Time}_\text{B}}{\text{Execution Time}_\text{A}} = n$$

   We say A is n times faster than B.

3. Breaking down execution time:

   (a) Factoring in cycle time:

$$\text{CPU time} = \text{CPU cycles} \times \text{cycle time}$$

   (b) How many cycles?

$$\text{CPU cycles} = \text{instruction count} \times \text{avg CPI}$$

   Categorize instructions and then get CPI for each category.

   How do we get instruction counts?

   CPU time:

$$\text{CPU time} = \text{instruction count} \times \text{avg CPI} \times \text{cycle time}$$

   Influences on:

   (a) Instruction count: compiler, architecture.

   Static vs. dynamic counts.

   (b) Cycle time: architecture, technology, microarchitecture (pipelining).

   (c) CPI: cycle time, microarchitecture (pipelining, superscalar, renaming).

   Complexity!!!

Examples:

1. Consider two different implementations, M1 and M2, of the same instruction set. There are four classes of instructions (A, B, C, and D) in the instruction set.

   M1 has a clock rate of 500 MHz. The average number of cycles for each instruction class on M1 is a follows:

Class	CPI
A	1
B	2
C	3
D	4

M2 has a clock rate of 750 MHz. The average number of cycles for each instruction class on M2 is a follows:

Class	CPI
A	2
B	2
C	4
D	4

Assume that peak performance is defined as the fastest rate that a machine can execute an instruction sequence chosen to maximize that rate. What are the peak performances of M1 and M2 expressed as instructions per second?

2. If the number of instructions executed in a certain program is divided equally among the classes of instructions, how much faster is M2 than M1?

3. Assuming the previous CPI and instruction distribution values, at what clock rate would M1 have the same performance as M2?

5 Choosing Benchmarks

1. How do you choose your test programs (benchmarks)?

   (a) Workload — programs used day-in and day-out?

   Too cumbersome. Everyone's workload differs.

   (b) Benchmarks — representative programs. Should be real, substantial applications.

   (c) **Not** synthetic kernels, ripe for one-shot compiler optimizations.

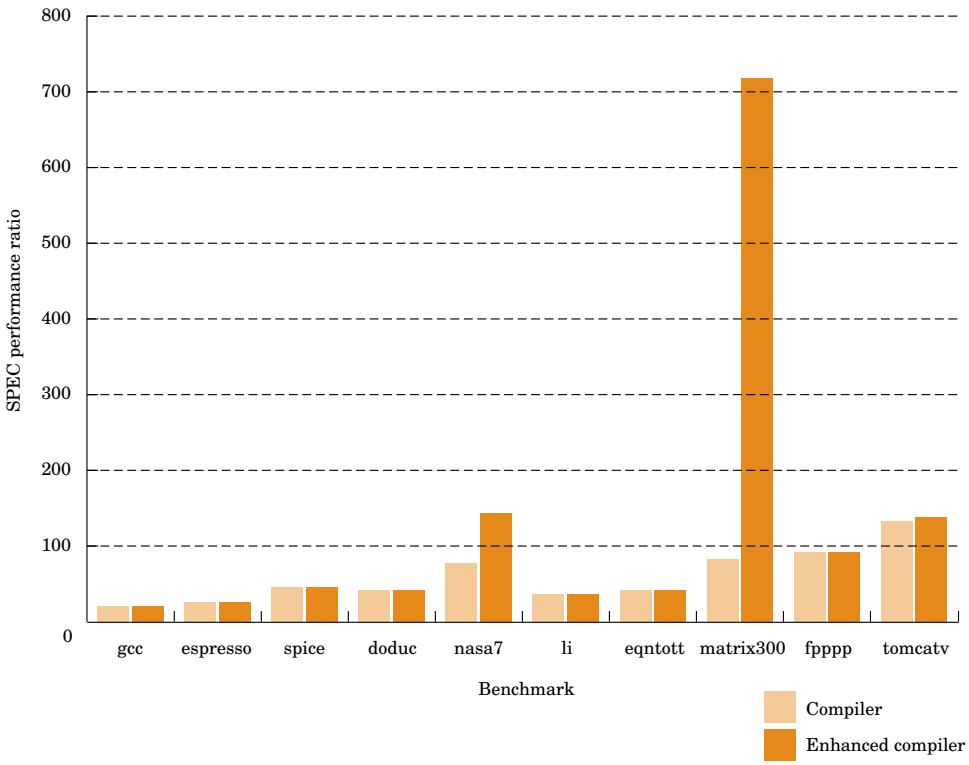
2. SPEC:

(a) Set of scientific benchmarks (compiler, go, compress, jpeg, plasma physics, quantum chemistry, etc.).

(b) '89 and '95.

(c) Int and fp.

3. Example of benchmark abuse: Matrix 300 (SPEC '89) on an IBM Powerstation 550:



99% of execution time is in a single line of code! Designed to test memory system. Compiler performed a one-shot optimization to eliminate cache misses.