Performance

Introduction

- Many factors impact performance:
- Technology:
  - basic circuit speed (clock speed, usually in MHz, now in GHz - billions of cycles per second)
  - process technology (# of transistors per chip)
- Organization:
  - what style of ISA (RISC vs. CISC)
  - what type of memory hierarchy
- Software: quality of compiler, OS, database, etc

Metrics

- Raw speed (peak performance -- never attained)
- *Execution time* (also called response time, ie. time required to execute program from beginning to end). Benchmarks:
  - Integer dominated programs (compilers, etc)
  - Scientific (lots of floating point)
  - Graphics/multimedia
- *Throughput* (total amount of work in given time)
  - Good metric for systems managers
  - Databases: keep the most people happy

Execution Time

Performance:

$\text{Performance}_A = 1/\text{ExecutionTime}_A$

Processor A is faster than Processor B if:

$\text{Performance}_A > \text{Performance}_B$

$\text{ExecutionTime}_A < \text{ExecutionTime}_B$

Relative Performance:

$\text{Performance}_A / \text{Performance}_B = \text{ExecutionTime}_B / \text{ExecutionTime}_A$

Measuring Execution Time

- Wall clock, response time, elapsed time
- Unix time function:

```
[fiji]:~ time someprogram
346.085u 0.39s 5:48.32 99.4% 5+202k 0+0io 0pf+0w
```

...lists user CPU time, system CPU time, elapsed time, percentage of elapsed time which is CPU time and other info

We'll typically use *User CPU time* to mean *CPU execution time*, or just *execution time*

Defining Execution Time

- Execution time = clock cycles x clock cycle time
- Execution time is program dependent
- Clock cycles are program dependent
- clock cycle time (usually in ns) is dependent on the machine

Since clock cycle time = 1/(clock cycle rate), and alternate definition is:

$$\text{CPU Execution time} = \frac{\text{CPU clock cycles}}{\text{clock cycle rate}}$$

CPI Cycles per Instruction

- Definition: CPI is the average # of cycles per instruction:
  - CPU clock cycles = Number of instructions executed x CPI

CPU Execution Time = Number of Instructions x CPI x clock cycle time

  - CPI in isolation is not a measure of performance (program and compiler dependent)
  - Ideally CPI = 1, but this might slow the clock (compromise)
  - Can we have CPI < 1

Instruction Classes

- We can have different CPIs for different classes of instructions (eg. floating point instructions take more cycles than integer instructions.)

CPU Execution time = Σ (CPI_i x C_i) x clock cycle time

  - C_i is the number of instructions in a class that have executed
  - Note that minimizing the number of instructions doesn't necessarily improve performance.
  - Improving part of the architecture can improve a C_i.

Measuring CPI

- Instruction count: need a simulator or profiler:
  - simulator interprets and counts each instruction
  - profiler uses a sampling technique
- CPU execution time can be measured
- Clock cycle time is given by processor
- We know Exetime, so we can solve for total cycles
- Knowing total cycles together with the number of instructions executed lets us solve for average CPI

Other Metrics: MIPS

- MIPS = Millions of Instructions Per Second

MIPS = Instruction count / (Execution Time x 1,000,000)

- MIPS is appealing because it is a rate -- bigger is better
- But MIPS in isolation is no better than CPI -- it's program dependent
- Does not take the instruction set into account:
  - CISC programs typically take fewer instructions than a RISC, so we can't compare the different ISAs using MIPS

The Trouble with MIPS

- It gives "wrong" results:
  - Machine A with compiler C1 executes program P in 10 seconds, using 100,000,000 instructions (10 MIPS)
  - Machine A with compiler C2 executes program P in 15 seconds, using 180,000,000 instructions (12 MIPS)
- C1 is clearly better, but it has a lower MIPS rating.
- MIPS doesn't take CPI into account...

Benchmarks

- Benchmark: workload representative of what the computer will be used for.
- CPU benchmarks: SPEC (SPECint, SPECfp, etc)
- Database benchmarks
- Webserver benchmarks
- Caveats:
  - Compilers optimize specifically for benchmarks
  - Some benchmarks don't test the memory system sufficiently

Amdahl's Law

- Amount we can improve performance is limited by the amount that the improved feature is actually used:

$$\text{New Execution Time} = \frac{\text{Execution Time affected by Improvement}}{\text{Amount of improvement}} + \text{Unaffected Exe time}$$

Example: if loads/stores take up 33% of our Exe time, how much do we need to improve loads/stores to make the program run 1.5 times faster?

Corollary: Make the common case fast!

Example Measurements

Category	GCC	SPICE	Ave CPI
Load/Store	33%	40%	1.4
Branches	16%	8%	1.8
Jumps	2%	2%	1.2
FP Add	-	5%	2.0
FP Sub	-	3%	4.0
FP Mul	-	6%	5.0
FP Div	-	3%	19.0
Other (integer ADD, etc)	49%	33%	1.0

- What is the average CPI for gcc? For spice?