Chapter 2: Performance and Cost

Focus on Cost/Performance Trade-offs

Performance Measures

- Elapsed Time vs. CPU Time
- Iron Law of Performance

Misleading Metrics

- MIPS
- MFLOPS

Benchmarks and Benchmarking

Averaging

Cost and Price

Performance Measures

$$\textit{Performance} = \textit{Rate} = \frac{1}{\textit{Time}}$$

Distinction is often blurred

Elapsed (wall-clock) Time

- Most important to users

CPU (virtual) time

- Most important to architects?

Iron Law of Performance

CPU time for a program can be factored into:

$$\textit{CPU time} = \left(\frac{\textit{instrns}}{\textit{program}}\right) \times \left(\frac{\textit{cycles}}{\textit{instrn}}\right) \times \left(\frac{\textit{time}}{\textit{cycle}}\right)$$

Instructions per program (path length)

- ISA and compiler

Cycles per instruction (CPI)

- ISA and organization (e.g., cache misses)
- As rate, called IPC

Time per cycle (cycle time, clock time)

- Organization and hardware
  (very hard to study in the abstract)
- Often a rate (e.g., 100 MHz ≡ 10 ns)

Controversial Example

$$\textit{CPU time} = \left(\frac{\textit{instrns}}{\textit{program}}\right) \times \left(\frac{\textit{cycles}}{\textit{instrn}}\right) \times \left(\frac{\textit{time}}{\textit{cycle}}\right)$$

Some have argued:

- CISC CPU time = $P \times 8 \times T = 8PT$
- RISC CPU time = $2P \times 2 \times T = 4PT$
- RISC CPU time = (CISC CPU time)/2

DISCLAIMER:

- The truth is much, much more complex

Standard Example: Load/Store Machine

Operation	Frequency	Cycle Count
ALU Ops	43%	1
Loads	21%	1
Stores	12%	2
Branches	24%	2

Suppose we could make stores execute in 1 cycle, by slowing down the cycle time by 15%.

Should we make this "optimization"?

Old CPI = 0.43 + 0.21 + (0.12 + 0.24)×2 = 1.36

New CPI = 0.43 + 0.21 + 0.12 + 0.24×2 = 1.24

$$\frac{\textit{New CPU time}}{\textit{Old CPU time}} = \frac{P \times \textit{New CPI} \times 1.15T}{P \times \textit{Old CPI} \times T} = 1.05$$

Conclusion: Don't make the change

MIPS: Meaningless Indicator of Performance

$$\textit{MIPS} = \left(\frac{\textit{instrns}}{\textit{program}}\right) \times \left(\frac{\textit{program}}{\textit{time}}\right) \times \left(\frac{1}{10^6}\right)$$

Depends upon instruction set

Instruction sets are not equivalent

Depends upon the program

Different programs use different instruction mixes

Can vary inversely with performance

e.g. Optimizing compilers

e.g. Adding floating-point hardware

People often ''factor out'' program

Relative MIPS

$$\textit{Relative MIPS} = \left(\frac{\textit{time}_{\textit{reference}}}{\textit{time}_{\textit{new}}}\right) \times \textit{MIPS}_{\textit{reference}}$$

Relative MIPS (e.g., VUPS or VAX MIPS) are a little better than native MIPS

However, very sensitive to reference machine.

E.g., VAX-11/780 has a small cache and inadequate write buffering.

Which compiler and operating system?

Change over time

MFLOPS: Just as Meaningless

$$\textit{MFLOPS} = \left(\frac{\textit{FP operations}}{\textit{program}}\right) \times \left(\frac{\textit{program}}{\textit{time}}\right) \times \left(\frac{1}{10^6}\right)$$

Depends upon program

Must be floating-point (FP) intensive

Some programs perform no FP operations

People often ''factor out'' program

PEAK MFLOPS: Performance the manufacturer guarantees you won't exceed

Normalized MFLOPS

Not all machines implement the same FP operations

- Cray-1 does not implement Divide
- Motorola 68882 does SQRT, SIN, and COS

Not all FP operations are the same

- ADD is much faster than Divide

Normalized MFOPS

- Assign a ‘‘canonical number of FP operations’’ to a program

$$\textit{normalized MFLOPS} = \left(\frac{\textit{canonical FP operations}}{\textit{time} \times 10^6} \right)$$

Benchmarks

Real Programs

- Representative of real workload
- The only accurate way to characterize performance

Kernels

- "Representative" program fragments
- Often only represent themselves

Toy Benchmarks

- e.g. Sieve, Puzzle, Towers
- NEVER, NEVER, NEVER!!!

Synthetic Benchmarks

- Worse than kernels
- Optimizing compilers wreck havoc

Benchmarking

Reproducible results

- Must control outside factors

Important factors

- Program input
- Version of program
- Version of compiler
- Optimization level
- Version of operating system
- Amount of memory
- Number and type of disks
- Version of CPU
- Cache Configuration

Benchmarking: SPEC

Limitations of de facto Benchmarks

Dhrystone

- Synthetic integer benchmark
- Heavy string emphasis
- Optimizing compilers cause MAJOR problems

Whetstone

- Synthetic floating-point benchmark
- Designed to thwart optimization

Linpack

- Floating-point kernel
- DAXPY() = $\boldsymbol{A}(\boldsymbol{I}) = \boldsymbol{B}(\boldsymbol{I}) + \boldsymbol{C} \times \boldsymbol{D}(\boldsymbol{I})$

Benchmarking: SPEC92

Integer

Espresso	CAD tool for PLAs
Li	lisp interpreter
Eqntott	equation translator
Compress	data compressor
Sc	speadsheet
Gcc	GNU C compiler

Floating-Point

Spice	circuit simulator
Doduc	Monte Carlo simulation
Mdljdp2	N-bodies w/ Lennard-Jones
Wave5	N-boides w/ Maxwell
Tomcatv	vectorized mesh generation
Ora	ray tracing
Alvinn	neural network
Ear	ear simulation w/ FFT
Mdljsp2	single-precision version of Mdljdp2
Swm256	finite differences for water
Su2cor	quantum physics
Hydor2s	galactic jets w/ Navier Stokes
Nasa7	NASA CFD kernels
Fpppp	quantumchemistry

Even so, don't believe too much

Summarizing Results: A Counter-Example

A car goes 30 MPH for the first ten miles and 90 MPH for the second ten miles. What the car's average speed over the twenty miles?

Wrong answer:

$$\textbf{\textit{Avg speed}} = \frac{30\ \textbf{\textit{MPH}} + 90\ \textbf{\textit{MPH}}}{2} = 60\ \textbf{\textit{MPH}}$$

Correct answer:

$$\textbf{\textit{Avg speed}} = \frac{\textbf{\textit{total distance}}}{\textbf{\textit{total time}}}$$

$$= \frac{10\ \textbf{\textit{miles}} + 10\ \textbf{\textit{miles}}}{(10\ \textbf{\textit{miles}})/(30\ \textbf{\textit{MPH}}) + (10\ \textbf{\textit{miles}})/(90\ \textbf{\textit{MPH}})}$$

$$= \frac{20\ \textbf{\textit{miles}}}{(1/3\ \textbf{\textit{hour}}) + (1/9\ \textbf{\textit{hour}})} = 45\ \textbf{\textit{MPH}}$$

Summarizing Results: Averages

Use the ARITHMETIC mean for times (cycles per instruction):

$$\frac{1}{n}\sum_{i=1}^{n} \textbf{\textit{time}}_i$$

Use the HARMONIC mean for rates (MIPS, MFLOPS):

$$\left(\frac{1}{n}\sum_{i=1}^{n} \frac{1}{\textbf{\textit{rate}}_i}\right)^{-1}$$

Use the GEOMETRIC mean for ratios (normalize numbers):

$$\left(\prod_{i=1}^{n} \textbf{\textit{ratio}}_i\right)^{1/n}$$

Better yet: don't average normalized numbers

Summarizing Results: A Measure of Time

Property 1:

> A single-number performance measure for a set of benchmarks expressed in ***units of time*** should be directly proportional to the total (weighted) time consumed by the benchmarks.

Property 2:

> A single-number performance measure for a set of benchmarks expressed as a ***rate*** should be indirectly proportional to the total (weighted) time consumed by the benchmarks.

Summarizing Results: Which Mean?

T_i = Execution time for Benchmark i

F_i = FP Operations for Benchmark i

$R_i = \frac{F_i}{T_i}$ = Rate for Benchmark i

Average Time:

$$A\text{-}mean = \frac{1}{n}\sum_{i=1}^{n} T_i$$

Average Rate:

$$A\text{-}mean = \frac{1}{n}\sum_{i=1}^{n} R_i$$

Violates Property 2: Not proportional to inverse of time Use Harmonic mean:

$$H\text{-}mean = \left(\frac{1}{n}\sum_{i=1}^{n}\frac{1}{R_i}\right)^{-1} = \left(\frac{1}{n}\sum_{i=1}^{n}\frac{F_i}{T_i}\right)^{-1}$$

Cost

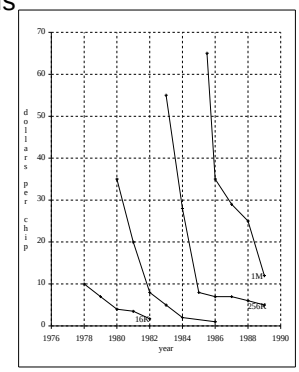
Cost is very important to most real designs

- But usually hard to quantify in the abstract

Costs change over time

- Learning curve lowers manufacturing costs
- Technology improvements lower costs

E.g. DRAMs



Integrated Circuit Cost

$$Cost\ of\ IC = \frac{Cost\ of\ Die + Cost\ of\ Testing + Cost\ of\ packaging}{Final\ Test\ Yield}$$

$$Cost\ of\ Die = \frac{Cost\ of\ Wafer}{Dies\ per\ Wafer \times Die\ Yield}$$

$$Die\ Yield = Wafer\ Yield \times \left\{1 + \frac{Defects\ per\ cm \times Die\ area}{\alpha}\right\}^{-\alpha}$$

Often $\alpha = 2.0$

$$Cost\ of\ Die = f\ (Die\ area^3)$$

Cost Breakdown

Component Cost

- Microprocessor, SRAM, DRAM
- Disk
- Power supplies and packaging

Direct Costs

- Manufacturing (labor, scrap)
- Warranty

Indirect Costs

- Research and Development
- Sales and Marketing
- General and Administrative
- Profits and Taxes

Price

Only loosely related to cost.

Start with all component costs

Add 25 - 40 % for direct costs (manufacturing, distribution, labor)

Add 46 - 65 % gross margin (R&D, marketing, general administration, profit)

= AVERAGE SELLING PRICE

Add 60 - 75 % for correct for discounts and allow dealer profit

= LIST PRICE

Note

Component cost 15 - 33 % of list price

R&D is often 8 - 15 % of list price

(C) 1994 by Hill, Wood & Sohi