Lecture 4: Benchmarks and Performance Metrics

Prof. Randy H. Katz
Computer Science 252
Spring 1996

Review

- **Designing to Last through Trends**

	Capacity	Speed
Logic	2x in 3 years	2x in 3 years
DRAM	4x in 3 years	1.4x in 10 years
Disk	4x in 3 years	1.4x in 10 years

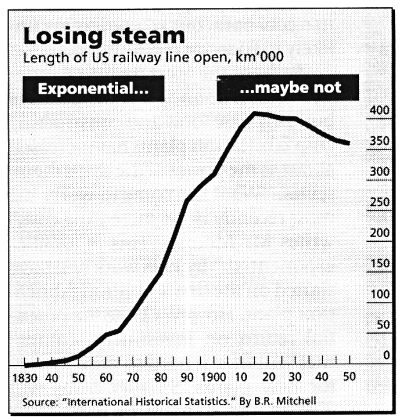
- **Time to run the task**
  - Execution time, response time, latency
- **Tasks per day, hour, week, sec, ns, …**
  - Throughput, bandwidth
- **"X is n times faster than Y" means**

$$\frac{\text{ExTime(Y)}}{\text{ExTime(X)}} = \frac{\text{Performance(X)}}{\text{Performance(Y)}}$$

The Danger of Extrapolation

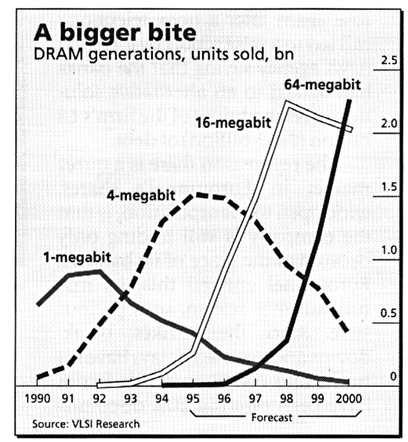
- Process today: 0.5 μm
- Limit of optical litho: 0.18 μm

- Power dissipation?
- Cost of new fabs?
- Alternative technologies?
  - GaAs
  - Optical



Doing Poorly by Doing Well

- Windows 95 drives huge demand for DRAM
- 16 Mbit chips not conveniently packaged for PCs (4 MByte SIMMs vs. 16 MByte SIMMs)
- 4 Mbit-by-4 vs. 1 Mbit-by-16



Aspects of CPU Performance

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

	Inst Count	CPI	Clock Rate
Program	X		
Compiler	X	(X)	
Inst. Set.	X	X	
Organization		X	X
Technology			X

Marketing Metrics

MIPS = Instruction Count / Time * 10^6 = Clock Rate / CPI * 10^6

- Machines with different instruction sets ?
- Programs with different instruction mixes ?
  - Dynamic frequency of instructions
- Uncorrelated with performance

MFLOP/s = FP Operations / Time * 10^6

- Machine dependent
- Often not where time is spent

Normalized:	
add,sub,compare,mult	1
divide, sqrt	4
exp, sin, . . .	8

Cycles Per Instruction

"Average cycles per instruction"

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

$$\text{CPU time} = \text{CycleTime} * \sum_{i=1}^{n} CPI_i * I_i$$

"Instruction Frequency"

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

Invest resources where time is spent!

Example: Calculating CPI

Base Machine (Reg / Reg)

Op	Freq	Cycles	CPI(i)	(% Time)
ALU	50%	1	.5	(33%)
Load	20%	2	.4	(27%)
Store	10%	2	.2	(13%)
Branch	20%	2	.4	(27%)
			1.5	

Typical Mix

Example

Add register / memory operations:
- One source operand in memory
- One source operand in register
- Cycle count of 2

Branch cycle count to increase to 3.

What fraction of the loads must be eliminated for this to pay off?

Base Machine (Reg / Reg)

Op	Freq	Cycles
ALU	50%	1
Load	20%	2
Store	10%	2
Branch	20%	2

Typical Mix

Example Solution

Exec Time = Instr Cnt x CPI x Clock

Op	Freq	Cycles	
ALU	.50	1	.5
Load	.20	2	.4
Store	.10	2	.2
Branch	.20	2	.3
Reg/Mem			
	1.00		1.5

Example Solution

Exec Time = Instr Cnt x CPI x Clock

Op	Freq	Cycles		Freq	Cycles	
ALU	.50	1	.5	.5 – X	1	.5 – X
Load	.20	2	.4	.2 – X	2	.4 – 2X
Store	.10	2	.2	.1	2	.2
Branch	.20	2	.3	.2	3	.6
Reg/Mem				X	2	2X
	1.00		1.5	1 – X		(1.7 – X)/(1 – X)

$$\frac{\text{Cycles}_{New}}{\text{Instructions}_{New}}$$

CPI_{New} must be normalized to new instruction frequency

RHK.S96 11

Example Solution

Exec Time = Instr Cnt x CPI x Clock

Op	Freq	Cycles		Freq	Cycles	
ALU	.50	1	.5	.5 – X	1	.5 – X
Load	.20	2	.4	.2 – X	2	.4 – 2X
Store	.10	2	.2	.1	2	.2
Branch	.20	2	.3	.2	3	.6
Reg/Mem				X	2	2X
	1.00		1.5	1 – X		(1.7 – X)/(1 – X)

$\text{Instr Cnt}_{Old} \times \text{CPI}_{Old} \times \text{Clock}_{Old} = \text{Instr Cnt}_{New} \times \text{CPI}_{New} \times \text{Clock}_{New}$

$1.00 \times 1.5 = (1 - X) \times (1.7 - X)/(1 - X)$

RHK.S96 12

Example Solution

Exec Time = Instr Cnt x CPI x Clock

Op	Freq	Cycles		Freq	Cycles	
ALU	.50	1	.5	.5 – X	1	.5 – X
Load	.20	2	.4	.2 – X	2	.4 – 2X
Store	.10	2	.2	.1	2	.2
Branch	.20	2	.3	.2	3	.6
Reg/Mem				X	2	2X
	1.00		1.5	1 – X		(1.7 – X)/(1 – X)

$\text{Instr Cnt}_{Old} \times \text{CPI}_{Old} \times \text{Clock}_{Old} = \text{Instr Cnt}_{New} \times \text{CPI}_{New} \times \text{Clock}_{New}$

$1.00 \times 1.5 = (1 - X) \times (1.7 - X)/(1 - X)$

$1.5 = 1.7 - X$

$0.2 = X$

ALL loads must be eliminated for this to be a win!

Programs to Evaluate Processor Performance

- (Toy) Benchmarks
  - 10-100 line program
  - e.g.: sieve, puzzle, quicksort
- Synthetic Benchmarks
  - Attempt to match average frequencies of real workloads
  - e.g., Whetstone, dhrystone
- Kernels
  - Time critical excerpts of real programs
  - e.g., Livermore loops
- Real programs
  - e.g., gcc, spice

Benchmarking Games

- Differing configurations used to run the same workload on two systems
- Compiler wired to optimize the workload
- Test specification written to be biased towards one machine
- Synchronized CPU/IO intensive job sequence used
- Workload arbitrarily picked
- Very small benchmarks used
- Benchmarks manually translated to optimize performance

Common Benchmarking Mistakes

- Only average behavior represented in test workload
- Skewness of device demands ignored
- Loading level controlled inappropriately
- Caching effects ignored
- Buffer sizes not appropriate
- Inaccuracies due to sampling ignored

Common Benchmarking Mistakes

- Ignoring monitoring overhead
- Not validating measurements
- Not ensuring same initial conditions
- Not measuring transient (cold start) performance
- Using device utilizations for performance comparisons
- Collecting too much data but doing too little analysis

SPEC: System Performance Evaluation Cooperative

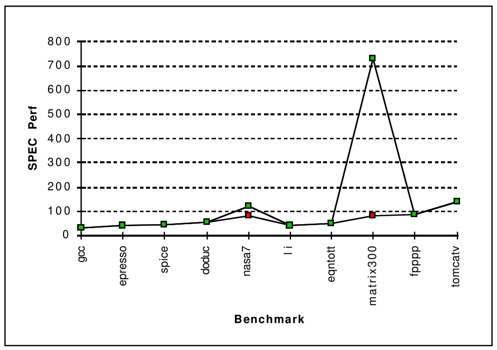
- **First Round 1989**
  - 10 programs yielding a single number
- **Second Round 1992**
  - SpecInt92 (6 integer programs) and SpecFP92 (14 floating point programs)
    - » Compiler Flags unlimited. March 93 of DEC 4000 Model 610:

```
spice: unix.c:/def=(sysv,has_bcopy,”bcopy(a,b,c)=
       memcpy(b,a,c)”
wave5: /ali=(all,dcom=nat)/ag=a/ur=4/ur=200
nasa7: /norecu/ag=a/ur=4/ur2=200/lc=blas
```

- **Third Round 1995**
  - Single flag setting for all programs; new set of programs “benchmarks useful for 3 years”

SPEC First Round

- One program: 99% of time in single line of code
- New front-end compiler could improve dramatically



How to Summarize Performance

- Arithmetic mean (weighted arithmetic mean) tracks execution time: $\sum(T_i)/n$ or $\sum(W_i*T_i)$
- Harmonic mean (weighted harmonic mean) of rates (e.g., MFLOPS) tracks execution time: $n/\sum(1/R_i)$ or $n/\sum(W_i/R_i)$
- Normalized execution time is handy for scaling performance
- But do not take the arithmetic mean of normalized execution time, use the geometric mean $(\prod(R_i)\wedge 1/n)$

Performance Evaluation

- Given sales is a function of performance relative to the competition, big investment in improving product as reported by performance summary
- Good products created when have:
  - Good benchmarks
  - Good ways to summarize performance
- If benchmarks/summary inadequate, then choose between improving product for real programs vs. improving product to get more sales;
  Sales almost always wins!
- Ex. time is the measure of computer performance!
- What about cost?