Lecture 4: Benchmarks and Performance Metrics

Prof. Randy H. Katz
Computer Science 252
Spring 1996
Review

• Designing to Last through Trends

<table>
<thead>
<tr>
<th>Capacity</th>
<th>Speed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logic</td>
<td>2x in 3 years</td>
</tr>
<tr>
<td>DRAM</td>
<td>4x in 3 years</td>
</tr>
<tr>
<td>Disk</td>
<td>4x in 3 years</td>
</tr>
</tbody>
</table>

• 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

![Graph showing historical data with a note on exponential growth and a caution against extrapolation.](source: "International Historical Statistics." By B.R. Mitchell)
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

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

<table>
<thead>
<tr>
<th></th>
<th>Inst Count</th>
<th>CPI</th>
<th>Clock Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compiler</td>
<td>X</td>
<td>(X)</td>
<td></td>
</tr>
<tr>
<td>Inst. Set.</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Organization</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Technology</td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Normalized</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>add, sub, compare, mult</td>
<td>1</td>
</tr>
<tr>
<td>divide, sqrt</td>
<td>4</td>
</tr>
<tr>
<td>exp, sin, ...</td>
<td>8</td>
</tr>
</tbody>
</table>
Cycles Per Instruction

“Average cycles per instruction”

\[
\text{CPI} = \frac{\text{Instruction Count}}{(\text{CPU Time} \times \text{Clock Rate})} = \frac{\text{Instruction Count}}{\text{Cycles}}
\]

CPU time = CycleTime \* \(\sum_{i=1}^{n} \text{CPI}_i \times I_i\)

“Instruction Frequency”

\[
\text{CPI} = \sum_{i=1}^{n} \text{CPI}_i \times F_i \quad \text{where} \quad F_i = \frac{I_i}{\text{Instruction Count}}
\]

Invest resources where time is spent!
Example: Calculating CPI

<table>
<thead>
<tr>
<th>Op</th>
<th>Freq</th>
<th>Cycles</th>
<th>CPI(i)</th>
<th>(% Time)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU</td>
<td>50%</td>
<td>1</td>
<td>.5</td>
<td>(33%)</td>
</tr>
<tr>
<td>Load</td>
<td>20%</td>
<td>2</td>
<td>.4</td>
<td>(27%)</td>
</tr>
<tr>
<td>Store</td>
<td>10%</td>
<td>2</td>
<td>.2</td>
<td>(13%)</td>
</tr>
<tr>
<td>Branch</td>
<td>20%</td>
<td>2</td>
<td>.4</td>
<td>(27%)</td>
</tr>
</tbody>
</table>

Typical Mix: \(\frac{1}{3} \)
### 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)**

<table>
<thead>
<tr>
<th>Op</th>
<th>Freq</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU</td>
<td>50%</td>
<td>1</td>
</tr>
<tr>
<td>Load</td>
<td>20%</td>
<td>2</td>
</tr>
<tr>
<td>Store</td>
<td>10%</td>
<td>2</td>
</tr>
<tr>
<td>Branch</td>
<td>20%</td>
<td>2</td>
</tr>
</tbody>
</table>

Typical Mix
## Example Solution

**Exec Time = Instr Cnt x CPI x Clock**

<table>
<thead>
<tr>
<th>Op</th>
<th>Freq</th>
<th>Cycles</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU</td>
<td>.50</td>
<td>1</td>
<td>.5</td>
</tr>
<tr>
<td>Load</td>
<td>.20</td>
<td>2</td>
<td>.4</td>
</tr>
<tr>
<td>Store</td>
<td>.10</td>
<td>2</td>
<td>.2</td>
</tr>
<tr>
<td>Branch</td>
<td>.20</td>
<td>2</td>
<td>.3</td>
</tr>
<tr>
<td>Reg/Mem</td>
<td>1.00</td>
<td>1.5</td>
<td></td>
</tr>
</tbody>
</table>
Example Solution

Exec Time = Instr Cnt \times CPI \times Clock

<table>
<thead>
<tr>
<th>Op</th>
<th>Freq</th>
<th>Cycles</th>
<th>Freq</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU</td>
<td>.50</td>
<td>1</td>
<td>.5</td>
<td>1</td>
</tr>
<tr>
<td>Load</td>
<td>.20</td>
<td>2</td>
<td>.4</td>
<td>2</td>
</tr>
<tr>
<td>Store</td>
<td>.10</td>
<td>2</td>
<td>.2</td>
<td>2</td>
</tr>
<tr>
<td>Branch</td>
<td>.20</td>
<td>2</td>
<td>.3</td>
<td>3</td>
</tr>
<tr>
<td>Reg/Mem</td>
<td>X</td>
<td>2</td>
<td>2X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1.00</td>
<td>1.5</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

CPI_{\text{New}} must be normalized to new instruction frequency

\[
\frac{\text{Cycles}_{\text{New}}}{\text{Instructions}_{\text{New}}} = \frac{(1.7 - X)/(1 - X)}{1 - X}
\]
## Example Solution

**Exec Time = Instr Cnt \times CPI \times Clock**

<table>
<thead>
<tr>
<th>Op</th>
<th>Freq</th>
<th>Cycles</th>
<th>Freq</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU</td>
<td>.50</td>
<td>1</td>
<td>.5 – X</td>
<td>1</td>
</tr>
<tr>
<td>Load</td>
<td>.20</td>
<td>2</td>
<td>.2 – X</td>
<td>2</td>
</tr>
<tr>
<td>Store</td>
<td>.10</td>
<td>2</td>
<td>.1</td>
<td>2</td>
</tr>
<tr>
<td>Branch</td>
<td>.20</td>
<td>2</td>
<td>.2</td>
<td>3</td>
</tr>
<tr>
<td>Reg/Mem</td>
<td>X</td>
<td>2</td>
<td>2X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1.00</td>
<td>1.5</td>
<td>1 – X</td>
<td>(1.7 – X)/(1 – X)</td>
</tr>
</tbody>
</table>

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

\[
1.00 \times 1.5 = (1 \times X) \times (1.7 \times X)/(1 \times X)
\]
Example Solution

Exec Time = Instr Cnt x CPI x Clock

<table>
<thead>
<tr>
<th>Op</th>
<th>Freq</th>
<th>Cycles</th>
<th>Freq</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU</td>
<td>.50</td>
<td>.5</td>
<td>.5 – X</td>
<td>1</td>
</tr>
<tr>
<td>Load</td>
<td>.20</td>
<td>.4</td>
<td>.2 – X</td>
<td>2</td>
</tr>
<tr>
<td>Store</td>
<td>.10</td>
<td>.2</td>
<td>.1</td>
<td>2</td>
</tr>
<tr>
<td>Branch</td>
<td>.20</td>
<td>.3</td>
<td>.2</td>
<td>3</td>
</tr>
<tr>
<td>Reg/Mem</td>
<td>X</td>
<td>2X</td>
<td>2X</td>
<td></td>
</tr>
</tbody>
</table>

Instr Cnt_{Old} x CPI_{Old} x Clock_{Old} = Instr Cnt_{New} x CPI_{New} x Clock_{New}

1.00 x 1.5 = (1 – X) x (1.7 – X)/(1 – X)

1.5 = 1.7 – X

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

![SPEC Perf Benchmark](chart.png)
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 \( (\prod(R_i)^{1/n}) \)
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?