Lecture Outline

- Performance Metrics
- Averaging
- Amdahl’s Law
- Benchmarks
- The CPU Performance Equation
  - Optimal Pipelining Case Study
- Modern Processor Analysis
- Begin ISAs…
Performance Metrics

- Execution Time is often what we target
- Throughput (tasks/sec) vs. latency (sec/task)
- How do we decide the tasks? Benchmarks
  - What the customer cares about, real applications
  - Representative programs (SPEC, SYSMARK, etc)
  - Kernels: Code fragments from real programs (Linpack)
  - Toy Programs: Sieve, Quicksort
  - Synthetic Programs: Just a representative instruction mix (Whetsone, Dhrystone)

Measuring Performance

- Total Execution Time:
  \[
  \frac{1}{n} \sum_{i=1}^{n} \text{Time}_i
  \]

  This is *arithmetic* mean
  - This should be used when measuring performance in execution *times* (CPI)
Measuring Performance

• Weighted Execution Time:

\[ \sum_{i=1}^{n} \text{Weight}_i \times \text{Time}_i \]

• What if P1 and P2 are not run equally?

Measuring Performance

• Normalized Execution Time

• Normalize to reference machine

• Can only use geometric mean (arithmetic mean can vary depending on the reference machine)

\[ \sqrt[n]{\prod_{i=1}^{n} \text{ExecutionTimeRatio}_i} \]

• Problem: Ratio not Execution Time is the result
Harmonic Mean: Motivation

- 30 mph for the first 10 miles
- 90 mph for the next 10 miles
- Average speed? \((30+90)/2 = 60\text{mph}\)

- WRONG! Average speed = total distance / total time
- \(20/(10/30+10/90) = 45\text{mph}\)

---

Harmonic Mean

- Each program has \(O\) operations
- \(n\) programs executed \(nO\) operations in \(\Sigma T_i\)
- Execution rate is then
  \[
  nO/ \Sigma T_i = n/ \Sigma (T_i/O) = n/ \Sigma 1/P_i
  \]
  where \(1/P_i\) is the rate of execution of program \(i\)

- Harmonic mean should be used when measuring performance in execution *rates* (IPC)
Amdahl’s Law
(Law of Diminishing Returns)

• Very Intuitive – Make the Common case fast

\[ \text{Speedup} = \frac{\text{Execution Time for task without enhancement}}{\text{Execution Time for task using enhancement}} \]

\[ \text{Execution time}_{\text{new}} = \text{Execution time}_{\text{old}} \times \left( \left(1 - \text{Fraction}_{\text{enhanced}}\right) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}} \right) \]

Amdahl’s Law Corollary

\[ \text{Speedup}_{\text{Overall}} = \frac{1}{\left( \left(1 - \text{Fraction}_{\text{enhanced}}\right) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}} \right)} \]

As \( \text{Speedup}_{\text{Enhanced}} \gg 0 \), \( \text{Speedup}_{\text{Overall}} = \frac{1}{\left(1 - \text{Fraction}_{\text{enhanced}}\right)} \)
Amdahl’s Law Example

Fraction_{enhanced}=95\%, \quad Speedup_{Enhanced}=1.1x

\text{Speedup}_{Overall}=1/((1-.95)+(.95/1.1))=1.094

Fraction_{enhanced}=5\%, \quad Speedup_{Enhanced}=10x

\text{Speedup}_{Overall}=1/((1-.05)+(.05/10))=1.047

Fraction_{enhanced}=5\%, \quad Speedup_{Enhanced}=\infty

\text{Speedup}_{Overall}=1/(1-.05)=1.052

Make the common case fast!

MIPS

- MIPS = \frac{\text{instruction count}}{(\text{execution time} \times 10^6)}
  = \frac{\text{clock rate}}{(\text{CPI} \times 10^6)}

- Problems
  - ISAs are not equivalent, e.g. RISC vs. CISC
    - 1 CISC instruction may equal many RISC!
  - Programs use different instruction mixes
  - May be ok when comparing same benchmarks, same ISA, same compiler, same OS

Computer Science 146
David Brooks
**MFLOPS**

- Same as MIPS, just FP ops
- Not useful either
  - FP-intensive apps needed
  - Traditionally, FP ops were slow, INT can be ignored
  - BUT, now memory ops can be the slowest!
- “Peak MFLOPS” is a common marketing fallacy
  - Basically, it just says #FP-pipes X Clock Rate

**GHz**

- Is this a metric? Maybe as good as the others…
- One number, no benchmarks, what can be better?
- Many designs are frequency driven

<table>
<thead>
<tr>
<th>Processor</th>
<th>Clock Rate</th>
<th>SPEC FP2000</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM POWER3</td>
<td>450 MHz</td>
<td>434</td>
</tr>
<tr>
<td>Intel PIII</td>
<td>1.4 GHz</td>
<td>456</td>
</tr>
<tr>
<td>Intel Pentium 4</td>
<td>2.4 GHz</td>
<td>833</td>
</tr>
<tr>
<td>Itanium-2</td>
<td>1.0 GHz</td>
<td>1356</td>
</tr>
</tbody>
</table>
Benchmark Suites

- SPEC CPU2000 (int and float) (Desktop, Server)
- EEMBC ("embassy"), SPECjvm (Embedded)
- TPC-C, TPC-H, SPECjbb, ECperf (Server)

### SPEC CPU2000: Integer Benchmarks

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Language</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>164.gzip</td>
<td>C</td>
<td>Compression</td>
</tr>
<tr>
<td>175.vpr</td>
<td>C</td>
<td>FPGA Circuit Placement and Routing</td>
</tr>
<tr>
<td>176.gcc</td>
<td>C</td>
<td>C Programming Language Compiler</td>
</tr>
<tr>
<td>181.mcf</td>
<td>C</td>
<td>Combinatorial Optimization</td>
</tr>
<tr>
<td>186.crafty</td>
<td>C</td>
<td>Game Playing: Chess</td>
</tr>
<tr>
<td>197.parser</td>
<td>C</td>
<td>Word Processing</td>
</tr>
<tr>
<td>252.eon</td>
<td>C++</td>
<td>Computer Visualization</td>
</tr>
<tr>
<td>253.perlbmk</td>
<td>C</td>
<td>PERL Programming Language</td>
</tr>
<tr>
<td>254.gap</td>
<td>C</td>
<td>Group Theory, Interpreter</td>
</tr>
<tr>
<td>255.vortex</td>
<td>C</td>
<td>Object-oriented Database</td>
</tr>
<tr>
<td>256.bzip2</td>
<td>C</td>
<td>Compression</td>
</tr>
<tr>
<td>300.twolf</td>
<td>C</td>
<td>Place and Route Simulator</td>
</tr>
</tbody>
</table>
**SPEC CPU2000: Floating Point Benchmarks**

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Language</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>168.wupwise</td>
<td>Fortran 77</td>
<td>Physics / Quantum Chromodynamics</td>
</tr>
<tr>
<td>171.swim</td>
<td>Fortran 77</td>
<td>Shallow Water Modeling</td>
</tr>
<tr>
<td>172.mgrid</td>
<td>Fortran 77</td>
<td>Multi-grid Solver: 3D Potential Field</td>
</tr>
<tr>
<td>173.applu</td>
<td>Fortran 77</td>
<td>Parabolic / Elliptic Partial Differential Equations</td>
</tr>
<tr>
<td>177.mesa</td>
<td>C</td>
<td>3-D Graphics Library</td>
</tr>
<tr>
<td>178.galgel</td>
<td>Fortran 90</td>
<td>Computational Fluid Dynamics</td>
</tr>
<tr>
<td>179.art</td>
<td>C</td>
<td>Image Recognition / Neural Networks</td>
</tr>
<tr>
<td>183.equate</td>
<td>C</td>
<td>Seismic Wave Propagation Simulation</td>
</tr>
<tr>
<td>187.facerec</td>
<td>Fortran 90</td>
<td>Image Processing: Face Recognition</td>
</tr>
<tr>
<td>188.ammp</td>
<td>C</td>
<td>Computational Chemistry</td>
</tr>
<tr>
<td>189.lucas</td>
<td>Fortran 90</td>
<td>Number Theory / Primality Testing</td>
</tr>
<tr>
<td>191.fma3d</td>
<td>Fortran 90</td>
<td>Finite-element Crash Simulation</td>
</tr>
<tr>
<td>200.sixtrack</td>
<td>Fortran 77</td>
<td>High Energy Nuclear Physics Accelerator Design</td>
</tr>
<tr>
<td>301.apsi</td>
<td>Fortran 77</td>
<td>Meteorology: Pollutant Distribution</td>
</tr>
</tbody>
</table>

**Server Benchmarks**

- **TPC-C (Online-Transaction Processing, OLTP)**

<table>
<thead>
<tr>
<th>System</th>
<th># / Processor</th>
<th>tpm</th>
<th>$/tpm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fujitsu PrimePower</td>
<td>128, 563MHz SPARC64</td>
<td>455K</td>
<td>$28.58</td>
</tr>
<tr>
<td>HP SuperDome</td>
<td>64, 875MHz PA8700</td>
<td>423K</td>
<td>$15.64</td>
</tr>
<tr>
<td>IBM p690</td>
<td>32, 1300MHz POWER4</td>
<td>403K</td>
<td>$17.80</td>
</tr>
</tbody>
</table>

- **TPC-H (Ad-hoc, decision support)**
CPU Performance Equation

- Execution Time = $\frac{\text{seconds}}{\text{program}}$

\[
\frac{\text{instructions}}{\text{program}} \times \frac{\text{cycles}}{\text{instruction}} \times \frac{\text{seconds}}{\text{cycle}}
\]

Program
Architecture (ISA)
Compiler

Compiler (Scheduling)
Organization (uArch)
Microarchitects

Technology
Physical Design
Circuit Designers

Common Architecture Trick

- Instructions/Program (Path-length) is constant
  - Same benchmark, same compiler
  - Ok usually, but for some ideas compiler may change
- Seconds/Cycle (Cycle-time) is constant
  - “My tweak won’t impact cycle-time”
  - Often a bad assumption
  - Current designs are ~15-20 FO4 Inverter Delays per cycle
- Just focus on Cycles/Instruction (CPI or IPC)
- Most academic architecture studies do just this!
CPU Performance Equation: Case Study

- Fundamental Architecture Decision: How do we choose the number of pipeline stages?

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Logic (FO4)</th>
<th>Latch (FO4)</th>
<th>Instruction Cycles</th>
<th>Seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single Stage</td>
<td>200</td>
<td>3</td>
<td>200</td>
<td></td>
</tr>
<tr>
<td>5 Stage CPU</td>
<td>200</td>
<td>15</td>
<td>5 stage</td>
<td></td>
</tr>
<tr>
<td>20 Stage CPU</td>
<td>200</td>
<td>60</td>
<td>20 stage</td>
<td></td>
</tr>
</tbody>
</table>

$$\frac{\text{instructions}}{\text{program}} \times \frac{\text{cycles}}{\text{instruction}} \times \frac{\text{seconds}}{\text{cycle}}$$

Measuring Instruction Counts and CPI

- Hardware counters on processors
- Instrumented execution (ATOM)
- Instruction-level interpretation (instruction counts)
- Execution-driven simulation
  - Detailed simulation of execution core and memory hierarchy
Pipelining Case Study

- Caveat: Fixed architecture sizings (only latencies vary)

Computer Science 146
David Brooks
For next time

- Some CISC vs. RISC commentary
- Multimedia ISAs
- Paper available on webpage
  - http://www.eecs.harvard.edu/cs146/
- Read through Ch1 and Ch2…
- Compiler/ISA overview, may start pipelining review

Computer Science 146
David Brooks