**Performance Overview**

- Execution time is the best measure of performance: simple, intuitive, straightforward.
- Two important quantitative methods:
  - Amdahl's Law and Speedup
  - CPI - cycles per instruction
- Benchmarks
- Metrics for summarizing performance data
- Pitfalls

**Time-based Metrics**

<table>
<thead>
<tr>
<th>Plane</th>
<th>DC to Paris</th>
<th>Speed</th>
<th>Passengers</th>
<th>Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boeing 747</td>
<td>6.5 hours</td>
<td>610 mph</td>
<td>470</td>
<td>286,700</td>
</tr>
<tr>
<td>Concorde</td>
<td>3 hours</td>
<td>1350 mph</td>
<td>132</td>
<td>178,200</td>
</tr>
</tbody>
</table>

- Elapsed time to run the task
  - Execution time, response time, latency
- Rate: tasks completed per day, hour, week, sec, ns
  - Throughput, bandwidth

**Amdahl's Law**

Suppose that enhancement E accelerates a fraction F of the task by a factor S, and the remainder of the task is unaffected.

<table>
<thead>
<tr>
<th>ExTime (X)</th>
<th>Performance (X)</th>
<th>Speedup (E)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExTime (Y)</td>
<td>Performance (Y)</td>
<td>Speedup (E)</td>
</tr>
</tbody>
</table>

**Performance comparisons**

"X is n times faster than Y" means

\[
\text{ExTime}(Y) \cdot \frac{\text{Performance}(X)}{\text{Performance}(Y)} = n
\]
Amdahl’s Law

- Floating point instructions improved to run 2X; but only 10% of actual instructions are FP

\[ \text{Speedup}_{\text{overall}} = \frac{1}{0.95} = 1.053 \]

Corollary: Make the common case fast!

Amdahl’s Law Corollary:

- All instructions require instruction fetch, only a fraction require data fetch/store.
  - \( \Rightarrow \) optimize instruction access over data access
- Access to small memories is faster.
  - \( \Rightarrow \) organize the storage hierarchy such that most frequent accesses are to the smallest/closest memory unit.
- Programs exhibit locality (spatial and temporal).
  - \( \Rightarrow \) implement pre-fetching of nearby code/data

CPU Time Analysis

Terminology
- IC = instruction count = number of instructions in the program
- CPI = cycles per instruction (varies for different instructions)
- clock cycle = length of time between clock ticks
  - Note: clock cycle = \( \frac{1}{\text{clock frequency}} \) where frequency is measured in MHz

If we assume the CPI is constant for all instructions, we have:

\[ \text{CPU time} = \text{IC} \times \text{CPI} \times \text{clock cycle} \]

More realistic CPU time analysis

- A given machine has several classes of instructions.
- Each class of instructions has its own cycle time.

This equation includes separate IC and CPI for each instruction class:

\[ \text{CPU time} = \sum \text{IC} \times \text{CPI} \times \text{clock cycle} \]

Alternatively, if we know the frequency of occurrence of each instruction type:

\[ \text{CPU time} = \text{IC} \times \sum \text{IC} \times \text{freq} \times \text{CPI} \times \text{clock cycle} \]

where \( \sum \text{IC} \times \text{freq} = \text{avg. IC} \)

Example: Calculating CPI

<table>
<thead>
<tr>
<th>Op</th>
<th>Freq</th>
<th>CPI</th>
<th>CPU/F</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 Miss

\[ \text{average CPI} \]

Speedup computation using the CPI eqn

\[ \text{Speedup} = \frac{\text{IC}(\text{old}) \times \text{CPI}(\text{old}) \times \text{clock cycle}(\text{old})}{\text{IC}(\text{new}) \times \text{CPI}(\text{new}) \times \text{clock cycle}(\text{new})} \]

When doing problems, identify which of the three components have changed (old \( \rightarrow \) new). Need only include those components in the speedup equation since unchanged ones will cancel out.
Programs to evaluate performance

- Real benchmark programs
- Kernels
- Toy benchmarks
- Synthetic benchmarks

SPEC Benchmark Suite (Standard Performance Evaluation Cooperative http://www.spec.org)

- First Round 1989
  - 10 programs yielding a single number ("SPECmarks")
- Second Round 1992
  - SPECint92 (8 integer programs) and SPECfp92 (14 floating point programs).
  - Compiler Flags unlimited
  - Normalized to VAX-11/780

SPEC Benchmark Suite - continued

- Third Round 1995 "benchmarks useful for 3 years"
  - SPECint95 (8 integer programs) and SPECfp95 (10 floating point)
  - Single flag setting for all programs: SPECint_base95, SPECfp_base95
- Fourth Round 2000
  - CINT2000 (12 integer programs) and CPF2000 (14 floating point)
  - usable across Unix and Windows NT
  - Normalized to Sun Ultra_10 workstation with 300-MHz SPARC and 256 MB memory
  - base and optimized versions
- Fifth Round 2006
  - HW problem !!

Additional desktop (PC) benchmarks

- Winbench
  - scripts that test CPU, video, and disk performance
- Business Winstone
  - netscape, office suite applications
- CC Winstone
  - content creation applications such as Photoshop

Specialized Benchmarks

- Graphics benchmarks SPECviewperf, SPECapc
- Embedded systems benchmarks EEMBC - automotive, consumer, networking, office automation, telecommunications
- Server benchmarks SPEC benchmarks for CPU, I/O system, web server, transaction servers (www.tpc.org)

How to Summarize Performance

Two metrics for summarizing execution time

- Arithmetic mean (weighted arithmetic mean) tracks execution time: \( \frac{\sum_{i} W_i T_i}{\sum_{i} W_i} \) or \( \frac{\sum_{i} W_i T_i}{\sum_{i} W_i} \)

- Harmonic mean (weighted harmonic mean) of rates (e.g., MFLOPS) tracks execution time: \( n \frac{\sum_{i} W_i}{\sum_{i} W_i R_i} \) or \( n \frac{\sum_{i} W_i}{\sum_{i} W_i R_i} \)
How to Summarize Performance

- Normalized execution time for scaling performance (e.g., X times faster than SPARCstation 10)
  - Arithmetic mean impacted by choice of reference machine

- Geometric mean for comparison of normalized execution times \( \prod (T_i)^{1/n} \)
  - Independent of chosen machine
  - but not good metric for total execution time

Fallacies and Pitfalls

- **Fallacy**: MIPS is an accurate measure of comparative performance.
  \[ \text{MIPS} = \frac{\text{instruction count}}{\text{execution time} \times 8^7} \times \frac{\text{clock rate}}{\text{CPI} \times 10^6} \]

- **Fallacy**: MFLOPS is a consistent and useful measure of performance.
  \[ \text{MFLOPS} = \frac{\text{floating point operations}}{\text{execution time} \times 10^6} \]

More Fallacies and Pitfalls

- **Fallacy**: Synthetic benchmarks predict performance for real programs.
- **Fallacy**: Benchmarks remain valid indefinitely.
- **Fallacy**: Peak performance tracks observed performance.
- **Fallacy**: Your performance in CIS 429/529 depends on how much you eat in class.