### 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

### Amdahl's Law

**Speedup due to enhancement E:**

\[
\text{Speedup}(E) = \frac{\text{ExTime w/o E}}{\text{ExTime w/ E}} = \frac{\text{Performance w/ E}}{\text{Performance w/o E}}
\]

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

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

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

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

\[
ExTime_{\text{new}} = ExTime_{\text{old}} \times (0.9 + 0.1/2) = 0.95 \times ExTime_{\text{old}}
\]

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

Corollary: Make the common case fast!
Amdahl’s Law Corollary: 
Make the common case fast!

• All instructions require instruction fetch, only a fraction require data fetch/store.
  ➞ optimize instruction access over data access
• Access to small memories is faster.
  ➞ organize the storage hierarchy such that most frequent accesses are to the smallest/closest memory unit.
• Programs exhibit locality (spatial and temporal).
  ➞ 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 = 1 / clock frequency
  where frequency is measured in MHz

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

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

\[
\text{CPU time} = \sum_{i} \text{CPI}_i \times \text{IC}_i \times \text{clock cycles}
\]

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

\[
\text{CPU time} = \text{IC} \times \left( \sum_{i} \text{CPI}_i \times \text{freq}_i \right) \times \text{clock cycles}
\]

where \( \sum_{i} \text{CPI}_i \times \text{freq}_i \) = avg. CPI

Example: Calculating CPI

<table>
<thead>
<tr>
<th>Op</th>
<th>Freq</th>
<th>CPI</th>
<th>CPI*Freq</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: 1.5

average CPI
Speedup computation using the CPI eqn

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

When doing problems, identify which of the three components have changed (old --> new). Need only include those components in the speedup equation since unchanged ones will cancel out.