Chapter 1

Computer Performance

Transistors

- Fun facts about 45nm transistors:
  - 30 million can fit on the head of a pin.
  - 2,000 fit across the width of a human hair.
  - If car prices had fallen at the same rate as the price of a single transistor has since 1968, a new car today would cost about 1 cent.
Chapter 1 — Computer Abstractions and Technology

Understanding Performance

- Algorithm
  - Determines the number of operations executed.
- Programming language, compiler, architecture
  - Determines the number of machine instructions executed per operation.
- Processor and memory system
  - Determines how fast instructions are executed.

Performance Metrics

- Possible measures:
  - Response time – elapsed time between start and end of a program (important to individual users).
  - Throughput – amount of work done in a fixed amount of time (important to data centers).
- The two measures are usually linked:
  - A faster processor will improve both.
  - Near-future processors will likely only improve throughput.
  - Some architecture improvements will improve throughput and worsen response time, like pipelining.
Speedup and Improvement

Example

1. What is the speedup of System X over System Y if System X executes a program in 10 seconds and system Y executes the same program in 15 seconds? 5 seconds or 1.5 times

2. What is the percentage reduction in execution time for the program of X compared to Y? (15-10)/15 = 33%

3. What is the percentage increase in execution time for the program of Y compared to X? (15-10)/10 = 50%

CPU Clocking

- Operation of digital hardware is governed by a constant-rate clock:

  - Clock frequency (rate) – cycles per second
    - e.g., 4.0GHz = 4000MHz = 4.0×10^9Hz
  - Clock period – duration of a clock cycle
    - e.g., 250ps = 0.25ns = 250×10^{-12}s
Performance Equation #1

\[ CPU \text{ execution time} = (CPU \text{ clock cycles})(\text{clock cycle time}) \]

\[ clock \text{ cycle time} = \frac{1}{\text{clock speed}} \]

- **Example #1:**
  - If a program runs for 10 seconds on a 3 GHz processor, how many clock cycles did it run for? 30 billion

- **Example #2:**
  - If a program runs for 2 billion clock cycles on a 1.5 GHz processor, what is the execution time in seconds? 1.333

Performance Equation #2

- **CPI** = Clock Cycles Per Instruction.

\[ cpu \text{ clock cycles} = (\text{number of instructions})(CPI) \]

- **Example:**
  - If a 2 GHz processor completes an instruction every third cycle, how many instructions are there in a program that runs for 10 seconds? \(10(2E9)/3 = 6.667E9\)
Performance Equation Summary

- Our basic performance equation is then:

\[ CPU \text{ time} = (\text{clock cycle time})(\text{instruction count})(CPI) \]

or

\[ CPU \text{ time} = \frac{(\text{instruction count})(CPI)}{\text{clock rate}} \]

- These equations separate the key factors that affect performance:
  - The CPU execution time is measured by running the program.
  - The clock rate is usually given.
  - The overall instruction count is measured by using profilers or simulators.
  - CPI varies by instruction type and the instruction set architecture.

Finding Average CPI

- Computing the overall effective CPI is done by looking at the different types of instructions and their individual cycle counts and averaging:

\[ \text{Overall effective CPI} = \sum_{i=1}^{n} (\text{CPI}_i \times \text{IC}_i) \]

- Where IC\textsubscript{i} is the count (percentage) of the number of instructions of class \textit{i} executed.
- CPI\textsubscript{i} is the (average) number of clock cycles per instruction for that instruction class.
- \( n \) is the number of instruction classes.
### Optimizing Example

<table>
<thead>
<tr>
<th>Op</th>
<th>Freq</th>
<th>CPI</th>
<th>Freq x CPI</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>5</td>
<td>.4 1.0 1.0</td>
</tr>
<tr>
<td>Store</td>
<td>10%</td>
<td>3</td>
<td>.3  .3  .3</td>
</tr>
<tr>
<td>Branch</td>
<td>20%</td>
<td>2</td>
<td>.4  .2  .4</td>
</tr>
<tr>
<td>Σ</td>
<td></td>
<td></td>
<td>2.2 1.6 2.0 1.95</td>
</tr>
</tbody>
</table>

- How much faster would the machine be if a better data cache reduced the average load time to 2 cycles?
  
  \[ \text{CPU time new} = 1.6 \times \text{IC} \times \text{CC} \quad \text{so} \quad 2.2/1.6 \text{ means } 37.5\% \text{ faster} \]

- How does this compare with using branch prediction to shave a cycle off the branch time?
  
  \[ \text{CPU time new} = 2.0 \times \text{IC} \times \text{CC} \quad \text{so} \quad 2.2/2.0 \text{ means } 10\% \text{ faster} \]

- What if two ALU instructions could be executed at once?
  
  \[ \text{CPU time new} = 1.95 \times \text{IC} \times \text{CC} \quad \text{so} \quad 2.2/1.95 \text{ means } 12.8\% \text{ faster} \]

### SPEC Benchmarking

- SPEC – System Performance Evaluation Corporation, an industry consortium that creates a collection of relevant programs.
  
  - The 2006 version includes 12 integer and 17 floating-point applications.
  - The SPEC rating specifies how much faster a system is, compared to a baseline machine – a system with SPEC rating of 600 is 1.5 times faster than a system with SPEC rating of 400.
  
  - Note that this rating incorporates the behavior of all 29 programs – this may not necessarily predict performance for your favorite program.
Benchmarking Performance

- Each vendor announces a SPEC rating for their system:
  - A measure of execution time for a fixed collection of programs.
  - It is a function of a specific CPU, memory system, IO system, operating system, compiler.
  - Enables easy comparison of different systems.
  - The key is coming up with a collection of relevant programs.

### CINT2006 for Intel Core i7 920

<table>
<thead>
<tr>
<th>Description</th>
<th>Name</th>
<th>Instruction Count x 10^9</th>
<th>CPI</th>
<th>Clock cycle time (seconds x 10^-9)</th>
<th>Execution Time (seconds)</th>
<th>Reference Time (seconds)</th>
<th>SPECratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interpreted string processing perl</td>
<td>2252</td>
<td>0.60</td>
<td>0.376</td>
<td>808</td>
<td>9770</td>
<td>19.2</td>
<td></td>
</tr>
<tr>
<td>Stock-sorting compression</td>
<td>bzip2</td>
<td>2300</td>
<td>0.70</td>
<td>0.376</td>
<td>629</td>
<td>9050</td>
<td>19.4</td>
</tr>
<tr>
<td>GNU C compiler</td>
<td>gcc</td>
<td>784</td>
<td>1.20</td>
<td>0.376</td>
<td>358</td>
<td>8050</td>
<td>22.5</td>
</tr>
<tr>
<td>Combinatorial optimization</td>
<td>mcf</td>
<td>221</td>
<td>2.86</td>
<td>0.376</td>
<td>221</td>
<td>9120</td>
<td>41.2</td>
</tr>
<tr>
<td>Go game (32)</td>
<td>go</td>
<td>1274</td>
<td>1.10</td>
<td>0.376</td>
<td>527</td>
<td>10400</td>
<td>19.9</td>
</tr>
<tr>
<td>Search game sequence</td>
<td>hmmner</td>
<td>2616</td>
<td>0.60</td>
<td>0.376</td>
<td>590</td>
<td>9330</td>
<td>15.8</td>
</tr>
<tr>
<td>Chess game (32)</td>
<td>sjeng</td>
<td>1948</td>
<td>0.80</td>
<td>0.376</td>
<td>588</td>
<td>12100</td>
<td>20.7</td>
</tr>
<tr>
<td>Quantum computer simulation</td>
<td>lbquantum</td>
<td>650</td>
<td>0.44</td>
<td>0.376</td>
<td>109</td>
<td>20720</td>
<td>190.0</td>
</tr>
<tr>
<td>Video compression</td>
<td>h264enc</td>
<td>3793</td>
<td>0.50</td>
<td>0.376</td>
<td>713</td>
<td>22130</td>
<td>31.0</td>
</tr>
<tr>
<td>Discrete event simulation library</td>
<td>omnetpp</td>
<td>367</td>
<td>2.10</td>
<td>0.376</td>
<td>290</td>
<td>8250</td>
<td>21.5</td>
</tr>
<tr>
<td>Game/3D path finding</td>
<td>aistar</td>
<td>1250</td>
<td>1.00</td>
<td>0.376</td>
<td>470</td>
<td>7020</td>
<td>14.0</td>
</tr>
<tr>
<td>XML parsing</td>
<td>xmlmindb</td>
<td>1945</td>
<td>0.70</td>
<td>0.376</td>
<td>276</td>
<td>6900</td>
<td>25.1</td>
</tr>
<tr>
<td>Geometric mean</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>25.7</td>
</tr>
</tbody>
</table>
Deriving a Single Performance Number

How is the performance of 29 different apps compressed into a single performance number?

- SPEC uses **Geometric Mean** (GM) – the execution time of each program is multiplied and the $N^{th}$ root is derived.
- Another popular metric is **Arithmetic Mean** (AM) – the average of each program’s execution time.
- Yet another is the **Weighted Arithmetic Mean** – the execution times of some programs are weighted to balance priorities.

Amdahl’s Law

- Architecture design is very bottleneck-driven – make the common case fast, do not waste resources on a component that has little impact on overall performance/power.
- Amdahl’s Law states that the performance improvement through an enhancement is limited by the fraction of time the enhancement comes into play:

$$T_{\text{improved}} = \frac{T_{\text{affected}}}{\text{improvement factor}} + T_{\text{unaffected}}$$
**Amdahl’s Law Example**

\[
T_{\text{improved}} = \frac{T_{\text{affected}}}{\text{improvement factor}} + T_{\text{unaffected}}
\]

- In a certain program, multiply instructions account for 80 seconds of the 100 second execution time. How much improvement in multiply is needed to double performance?

\[
50 = \frac{80}{n} + 20
\]

\[n = 8/3\]

**Common Principles for Computers**

- Make the common case fast.
- Principle of locality
  - The same data/code will be used again (temporal locality).
  - Nearby data/code will be used next (spatial locality).
- Energy
  - Systems use energy even when idle.
- 90/10 rule – 10% of the program accounts for 90% of the execution time.
- Amdahl’s Law.
Chapter 1 Recap

- Knowledge of hardware improves software quality – compilers, OS, threaded programs, memory management.
- Important trends to follow:
  - Transistor sizing.
  - Move to multi-core.
  - Slowing rate of performance improvement.
  - Power/thermal constraints.
  - Long memory/disk latencies.
- Reasoning about performance – clock speeds, CPI, benchmark suites, performance equations.
- Next class period – MIPS architecture.