Lecture Topics

- Measuring, Reporting and Summarizing Performance
  - Execution Time and Throughput
  - Benchmarks
  - Comparing and Summarizing Performance Results
  - Processor Performance Equation

Reference: Chapter 1: Sections 1.8 (Pages 36 – 44)

Measuring and Reporting Performance

- User’s Perspective
  - A computer is faster when a program/task runs in less time
  - *Response time or execution time*
    - The time between the start and the completion of a task

- Amazon.com Administrator’s Perspective
  - A computer is faster when it completes more tasks per unit time (e.g., transactions serviced per hour)
  - *Throughput*
    - Total amount of work done in a given time

Which is Faster?

<table>
<thead>
<tr>
<th>Plane</th>
<th>DC to Paris</th>
<th>Speed</th>
<th>Passengers</th>
<th>Throughput (pmph)</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>

- From the perspective of an airline passenger
  - Concorde may be better (shorter flight time)

- From the perspective of the airline
  - Boeing may be better (more passengers per day)
Measuring and Reporting Performance

• “Computer X is n times faster than computer Y” means:

\[
\text{Speedup (n)} = \frac{\text{Execution time of Y}}{\text{Execution time of X}}
\]

\[
\% \text{ Improvement} = 1 - \frac{1}{\text{Speedup}}
\]

e.g., if X and Y complete a task in 20 sec. and 40 sec. respectively, then speedup = 40/20 = 2 and % Improvement = 1 – ½ = 50%

Measuring and Reporting Performance

• We might say “The throughput of server X is n times higher than server Y”:

\[
\frac{\text{Execution time of Y}}{\text{Execution time of X}} = \frac{\text{Tasks executed per second by X}}{\text{Tasks executed per second by Y}}
\]

or

\[
\frac{\text{Throughput of X}}{\text{Throughput of Y}}
\]

How to Evaluate Performance

• Simulate a mixture of workloads which we hope to be representative of a typical user and therefore predictive of the performance as perceived by the user:
  – Real applications
  – Modified (or scripted) applications
  – Kernels
  – Toy benchmarks
  – Synthetic benchmarks

Measuring and Reporting Performance

• The most meaningful way to measure response time is by elapsed time or wall clock time which accounts for:
  – Disk accesses
  – Memory accesses
  – I/O activities
  – Other users (unless system is unloaded)
  – CPU time
    • User time
    • OS time (on behalf of user)
Benchmark Suites

- **SPEC**
  - Standard Performance Evaluation Corporation
  - [www.spec.org](http://www.spec.org)
  - Benchmarks for wide variety of application classes
  - Mimic real programs

- **Desktop Benchmarks**
  - CPU Intensive
    - SPEC CPU 2006
      - 12 integer benchmarks: CINT2006
      - 17 floating point benchmarks: CFP2000
  - Graphics Intensive
    - SPECviewperf
      - 3D rendering

- **Server Benchmarks**
  - Variety in application and therefore benchmarks
  - Higher I/O activity (network, file I/O) than desktop benchmarks
  - SPEC
    - Multiple copies of SPECcpu benchmarks: SPECrate
    - File server benchmark: SPECCSFS
    - Web server benchmark: SPECWeb
    - Virtualized data-center servers: SPECvirt_sc2010
  - TPC (Transaction Processing Council)
    - Complex query environment: TPC-C
    - Warehouse transaction: TPC-W
    - Measure is “transactions/second”
  - Throughput-oriented but under some response time limits

Benchmark Suites

- **Embedded Benchmarks**
  - Difficulties
    - Extremely diverse application area
    - Simple appliance control to high performance networking switch
    - May execute only one program for entire product life
  - EEMBC
    - EDN Embedded Microprocessor Benchmark Consortium
    - Five Classes
      - Automotive/Industrial
      - Consumer
      - Networking
      - Office Automation
      - Telecommunications

Comparing and Summarizing Performance

Execution times of two programs on three systems

<table>
<thead>
<tr>
<th></th>
<th>Computer A</th>
<th>Computer B</th>
<th>Computer C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program P1</td>
<td>1</td>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>Program P2</td>
<td>1000</td>
<td>100</td>
<td>20</td>
</tr>
<tr>
<td>Total Time</td>
<td>1001</td>
<td>110</td>
<td>40</td>
</tr>
</tbody>
</table>

A is 10x faster than B for program P1
B is 10x faster than A for program P2
A is 20x faster than C for program P1
C is 50x faster than A for program P2
B is 2x faster than C for program P1
C is 5x faster than B for program P2

How to summarize these performance results?
Comparing and Summarizing Performance

- **Total Execution Time:**
  - A Consistent Summary Measure

<table>
<thead>
<tr>
<th>(Time in secs)</th>
<th>Computer A</th>
<th>Computer B</th>
<th>Computer C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program P1</td>
<td>1</td>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>Program P2</td>
<td>1000</td>
<td>100</td>
<td>20</td>
</tr>
<tr>
<td>Total Time</td>
<td>1001</td>
<td>110</td>
<td>40</td>
</tr>
</tbody>
</table>

Conclude:
- B is 9.1x faster than A for programs P1 and P2
- C is 25x faster than A for programs P1 and P2
- C is 2.75x faster than B for programs P1 and P2

**Weighting and Arithmetic Mean**

<table>
<thead>
<tr>
<th>Computers</th>
<th>Weightings</th>
<th>Weightings</th>
<th>Weightings</th>
<th>Weightings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program P1 (secs)</td>
<td>1</td>
<td>10</td>
<td>0.5</td>
<td>0.9</td>
</tr>
<tr>
<td>Program P2 (secs)</td>
<td>100</td>
<td>19</td>
<td>0.5</td>
<td>0.1</td>
</tr>
</tbody>
</table>

- Arithmetic Mean: W(1) = 50.5
- Arithmetic Mean: W(2) = 10.9
- Arithmetic Mean: W(3) = 1.99

Different choices of weights can lead to different conclusions
No fair choice of weights for performance comparisons

**Solution: Normalizing Execution Times**

- Normalize execution times to a reference system
  - Divide the execution time on the computer being rated by the execution time on the reference computer

  \[
  \text{Normalized time on processor } A = \frac{\text{Execution time}_A}{\text{Execution time}_\text{reference}}
  \]

  \[
  \text{Normalized time on processor } B = \frac{\text{Execution time}_B}{\text{Execution time}_\text{reference}}
  \]

- Choice of reference computer irrelevant when comparisons are made as a ratio

\[
\text{Speedup of B compared with A} = \frac{\text{ExecTime}_A/\text{ExecTime}_{B\text{reference}}}{\text{ExecTime}_A/\text{ExecTime}_{B\text{reference}}} = \frac{\text{ExecTime}_A}{\text{ExecTime}_B}
\]
Performance Comparison Methodology

To compare the performance of computer B with computer A
• **Step 1:** Pick a reference computer
• **Step 2:** For each program in the benchmark suite:
  – compute the normalized execution times for both computer A and computer B
• **Step 3:** Take the means of the normalized execution times for both computer A and computer B

• Which type of mean to use?
  – Arithmetic means: NO
    • Ratios have no units, arithmetic mean is meaningless
    • Skewed by size of input
      – Benchmark with best differential performance dominates others
  – Geometric Mean: YES

<table>
<thead>
<tr>
<th>Normalized Execution Time</th>
<th>Normalized to A</th>
<th>Normalized to B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program P1</td>
<td>1.0</td>
<td>10.0</td>
</tr>
<tr>
<td>Program P2</td>
<td>1.0</td>
<td>0.01</td>
</tr>
<tr>
<td>Geometric Mean</td>
<td>1.0</td>
<td>0.316</td>
</tr>
</tbody>
</table>

Speedup of B compared with A = $\frac{1}{0.316} = \frac{3.16}{1} = 3.16$

Relative geometric means consistent regardless of the reference system

Processor Performance Equation

$CPU \ time = Clock \ cycles \ for \ a \ program \times Clock \ cycle \ time$

or $CPU \ time = \frac{Clock \ cycles \ for \ a \ program}{Clock \ rate}$

$Clock \ cycles \ for \ a \ program = Instruction \ count \times Average \ # \ of \ cycles \ per \ instruction$

$= Instruction \ count \times CPI$

where CPI is the average number of clock cycles per instruction

Processor Performance Equation

$CPU \ time = \frac{Instruction \ count \times CPI}{Clock \ Rate}$

Or

$CPU \ time = \frac{Seconds}{Program} \times \frac{Instructions}{Program} \times \frac{Clock \ cycles}{Instruction} \times \frac{Seconds}{Clock \ cycle}$
Performance Equation Parameters

• **Clock rate**
  – Instruction’s execution divided into multiple steps, each step 1 clock cycle long
  – Clock rate is equal to number of clock cycles per second (Hertz or Hz)
  – Clock period is equal to the length of clock cycle (clock rate = 1 / clock period)

• **Cycles per Instruction (CPI)**
  – Avg. no. of clock cycles required to execute an instruction
    • For ALU instructions, CPI depends on circuit speed and logic complexity
    • For memory instructions, CPI depends on memory access latency

• **Number of Instructions**
  – Count of *dynamic* instructions required to complete a program
    • Depends on the instruction set architecture
  – Not to be confused with code size (no. of instructions in a source program)
    • For example, 8-instruction loop executing 5 times counted as 40 instructions

How to Increase Performance?

• **Faster logic and memory**
  – Design faster implementations of arithmetic and logic circuits
  – Use caches to reduce average latency of memory operations

• **Increase parallelism**
  – Perform multiple instructions in parallel (*instruction-level parallelism*)
    • Fetch next instruction while ALU executes previous instruction (*pipelining*)
    • Use multiple ALUs to execute multiple instructions concurrently (*superscalar*)
    • Result: CPI can become < 1
  – Execute multiple programs at the same time (*thread-level parallelism*)
    • Multiple cores on a single die (*multi/many-core processors*)

• **Choice of Instruction Set Architecture (ISA)**
  – Tradeoff between *Instruction count*, CPI and/or Clock Rate
  – **Complex Instruction Set Computers (CISC)**
    • Fewer, more complex instructions per program (lower *instruction count*)
    • but higher CPI and/or lower clock rate
  – **Reduced Instruction Set Computers (RISC)**
    • Simpler instructions, more needed to perform the same task (higher *instruction count*)
    • but instructions quicker to execute (lower CPI and/or higher clock rate)

• **Better compiler technology**
  – Advances in manufacturing technology
    • Faster transistors => Faster circuits => Less time needed per clock cycle
  – Reducing the amount of processing done in each step (*deeper pipeline*)
    • This may increase CPI, unless there is instruction-level parallelism
  – Converts a high-level program into least expensive set of machine instructions with the goal of reducing “Instruction count * CPI”
  – Optimized according to a specific processor architecture

How to Increase Performance?