Measuring and Evaluating Computer System Performance

The bottom line: Performance

<table>
<thead>
<tr>
<th>Car</th>
<th>Time to Bay Area</th>
<th>Speed</th>
<th>Passengers</th>
<th>Throughput (pmpm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ferrari</td>
<td>3.1 hours</td>
<td>160 mph</td>
<td>2</td>
<td>320</td>
</tr>
<tr>
<td>Greyhound</td>
<td>7.7 hours</td>
<td>65 mph</td>
<td>60</td>
<td>3900</td>
</tr>
</tbody>
</table>

° Time to do the task
  – execution time, response time, latency
° Tasks per day, hour, week, sec, ns. ..
  – throughput, bandwidth

How to measure Execution Time?

% time program
... program results ...
90.7u 12.9s 2:39 65%
%

- Wall-clock time?
- user CPU time?
- user + kernel CPU time?
Answer:
Our definition of Performance

\[
\text{Performance}_x = \frac{1}{\text{Execution Time}_x}, \text{ for program X}
\]

- only has meaning in the context of a program or workload
- Not very intuitive as an absolute measure, but most of the time we’re more interested in relative performance.

Relative Performance

- can be confusing
  - A runs in 12 seconds
  - B runs in 20 seconds
  - \( A/B = 0.6 \), so A is 40% faster, or 1.4X faster, or B is 40% slower
  - \( B/A = 1.67 \), so A is 67% faster, or 1.67X faster, or B is 67% slower
- needs a precise definition

Relative Performance, the Definition

\[
\text{Relative Performance}\ _{(X/Y)} = \frac{\text{Performance}_x}{\text{Performance}_y} = \frac{\text{Execution Time}_y}{\text{Execution Time}_x} = n
\]

"X is n times faster than Y"
"X is n times as fast as Y"
"From Y to X, speedup is n"

Example

- Machine A runs program C in 9 seconds, Machine B runs the same program in 6 seconds. What is the speedup we see if we move to Machine B from Machine A?
- Machine B gets a new compiler, and can now run the program in 3 seconds. ???
What is Time?

CPU Execution Time = CPU clock cycles * Clock cycle time

- Every conventional processor has a clock with an associated clock cycle time or clock rate
- Every program runs in an integral number of clock cycles

Cycle Time
- MHz = millions of cycles/second, GHz = billions of cycles/second
- X MHz = 1000/X nanoseconds cycle time
- Y GHz = 1/Y nanoseconds cycle time

How many clock cycles?

Number of CPU cycles = Instructions executed * Average Clock Cycles per Instruction (CPI)

Computer A runs program C in 3.6 billion cycles. Program C consists of 2 billion dynamic instructions. What is the CPI?

How many clock cycles?

Number of CPU cycles = Instructions executed * Average Clock Cycles per Instruction (CPI)

A computer is running a program with CPI = 2.0, and executes 24 million instructions, how long will it run?

All Together Now

CPU Execution Time = Instruction Count \times CPI \times Clock Cycle Time
Who Affects Performance?

- programmer
- compiler
- instruction-set architect
- machine architect
- hardware designer
- materials scientist/physicist/silicon engineer

Performance Variation

<table>
<thead>
<tr>
<th>Number of instructions</th>
<th>CPU Time</th>
<th>CPI</th>
<th>Clock Cycle Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Same machine different programs</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>same programs, different machines, same ISA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Same programs, different machines</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
MIPS

MIPS = Millions of Instructions Per Second
    = Instruction Count
    \frac{\text{Execution Time} \times 10^9}{\text{Clock rate}}
    = CPI \times 10^9

- Program-independent?
- Deceptive

FLOPS

FLOPS = FLoating-point Operations Per Second

- Program-independent?
  - Which operations?
- Useful, sometimes
  - "Theoretical peak" FLOPS, peak FLOPS, sustained FLOPS
- How does execution time depend on FLOPS?

Which Programs?

- peak throughput measures (simple programs)?
- synthetic benchmarks (whetstone, dhrystone,...)?
- "kernels" of useful computation (lapack, fftw, ...)
- Real applications
- SPEC (best of both worlds, but with problems of their own)
  - System Performance Evaluation Cooperative
  - Provides a common set of real applications along with strict guidelines for how to run them.
  - provides a relatively unbiased means to compare machines.

Danger in Benchmark-Specific Performance Measures

- measures compiler as much as architecture
  - (what about kernels?)
SPEC Performance on Pentium III and Pentium 4

Amdahl’s Law

- The impact of a performance improvement is limited by the percent of execution time affected by the improvement

\[
\text{Execution time after improvement} = \frac{\text{Execution Time Affected}}{\text{Amount of Improvement}} + \text{Execution Time Unaffected}
\]

- Make the common case fast!!

Key Points

- Be careful how you specify performance
- Execution time = instructions * CPI * cycle time
- Use real applications
- Use standards, if possible
- Make the common case fast