CSE 141 – Computer Architecture
Fall 2005

Lecture 5
Performance

Pramod V. Argade
October 10, 2005
Announcements

- **Reading Assignment**
  - Chapter 4. Assessing and Understanding Performance
    Sections 4.1 - 4.6

- **Homework 3: Due Mon., October 17th in class**
  4.1, 4.2, 4.6, 4.7, 4.8, 4.11, 4.12, 4.19, 4.22, 4.45

  **Additional Problem for Chapter 2:** Write a program in C or C++ to declare and initialize an array of two “ints.” Print a word at a non-word aligned address starting in the first element of the array. Run the program on a PC and on a SPARC Station. (**Hint:** use a pointer to access the word.)

  A. What is the output on each computer?
  B. If the program behavior is different on the two computers give explanation.

- **Quiz**
  **When:** Mon., October 17th, First 10 minutes of the class
  **Topic:** Performance, Chapter 4  **Need:** Paper, pen
# Course Schedule

<table>
<thead>
<tr>
<th>Lecture #</th>
<th>Date</th>
<th>Day</th>
<th>Lecture Topic</th>
<th>Quiz Topic</th>
<th>Homework Due</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>9/26</td>
<td>Monday</td>
<td>Introduction, Ch. 1</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>9/28</td>
<td>Wednesday</td>
<td>ISA, Ch. 2</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>10/3</td>
<td>Monday</td>
<td>Arithmetic Part 1, Ch. 4</td>
<td>ISA</td>
<td>#1</td>
</tr>
<tr>
<td>4</td>
<td>10/5</td>
<td>Wednesday</td>
<td>Arithmetic Part 2, Ch. 4</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>5</td>
<td>10/10</td>
<td>Monday</td>
<td>Performance, Ch. 3</td>
<td>Arithmetic</td>
<td>#2</td>
</tr>
<tr>
<td>6</td>
<td>10/12</td>
<td>Wednesday</td>
<td>Single cycle CPU, Ch. 5</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td>10/17</td>
<td>Monday</td>
<td>Single cycle CPU, Ch. 5</td>
<td>Performance</td>
<td>#3</td>
</tr>
<tr>
<td>8</td>
<td>10/19</td>
<td>Wednesday</td>
<td>Multi-cycle CPU, Ch. 5</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>9</td>
<td>10/24</td>
<td>Monday</td>
<td>Multi-cycle CPU, Ch. 5</td>
<td>Single Cycle CPU</td>
<td>#4</td>
</tr>
<tr>
<td>10</td>
<td>10/26</td>
<td>Wednesday</td>
<td>Review for the Midterm</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>10/31</td>
<td>Monday</td>
<td>Mid-term Exam</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>11/2</td>
<td>Wednesday</td>
<td>Exceptions, Ch. 5 and Pipelining, Ch. 6</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>12</td>
<td>11/7</td>
<td>Monday</td>
<td>Pipelining, Ch. 6</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>13</td>
<td>11/9</td>
<td>Wednesday</td>
<td>Data and control hazards, Ch. 6</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>14</td>
<td>11/14</td>
<td>Monday</td>
<td>Data and control hazards, Ch. 6</td>
<td>Pipeline Hazards</td>
<td>#5</td>
</tr>
<tr>
<td>15</td>
<td>11/16</td>
<td>Wednesday</td>
<td>Memory &amp; cache design, Ch. 7</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>16</td>
<td>11/21</td>
<td>Monday</td>
<td>Memory &amp; cache design, Ch. 7</td>
<td>Cache</td>
<td>#6</td>
</tr>
<tr>
<td>17</td>
<td>11/23</td>
<td>Wednesday</td>
<td>Virtual Memory &amp; cache design, Ch. 7</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>18</td>
<td>11/28</td>
<td>Monday</td>
<td>Course Review</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>12/8</td>
<td>Thursday</td>
<td>Final Exam 7 - 10 PM</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>
Performance Assessment

<table>
<thead>
<tr>
<th>Airplane</th>
<th>Passenger Capacity</th>
<th>Cruising Range (miles)</th>
<th>Cruising Speed (m.p.h.)</th>
<th>Passenger Throughput (passenger x m.p.h.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boeing 777</td>
<td>375</td>
<td>4630</td>
<td>610</td>
<td>228,750</td>
</tr>
<tr>
<td>Boeing 747</td>
<td>470</td>
<td>4150</td>
<td>610</td>
<td>286,700</td>
</tr>
<tr>
<td>Douglas DC-8-50</td>
<td>146</td>
<td>8720</td>
<td>544</td>
<td>79,424</td>
</tr>
<tr>
<td>Airbus A380</td>
<td>555</td>
<td>8000</td>
<td>645</td>
<td>357,975</td>
</tr>
</tbody>
</table>

- Example definitions of performance
  - Passengers * MPH
  - Cruising speed
  - Passenger capacity
  - (Passenger * MPH)/$ ⇐ need more data!

- Clearly formulate your definition of performance
Why worry about performance?

- Learn to measure, report, and summarize
- Make intelligent choices
- See through the marketing hype
- As a designer or purchaser, assess:
  - which system has best performance?
  - which system has lowest cost?
  - which system has highest performance/cost?
- Formulate metrics for performance measurement
  - How to report relative performance?
- Understand impact of architectural choices on performance
Computer Performance: TIME, TIME, TIME

- Response Time (latency)
  - How long does it take for my job to run?
  - How long does it take to execute a job?
  - How long must I wait for the database query?

- Throughput
  - How many jobs can the machine run at once?
  - What is the average execution rate?
  - How much work is getting done?

- *If we upgrade a machine with a new processor what do we decrease?*

- *If we add a new machine to the lab what do we increase?*
Execution Time

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

- Elapsed Time (2:39)
  - Counts everything *(disk and memory accesses, I/O, etc.)*
  - A useful number, but often not good for comparison purposes

- Total CPU time (103.6 s)
  - Doesn't count I/O or time spent running other programs
  - Comprised of system time (12.9 s), and user time (90.7 s)

- Percent of time spent by CPU running this program (65%)

- Our focus: user CPU time
  - Time spent executing the lines of code that are "in" our program
A Definition of Performance

- For some program running on machine X,
  \[ \text{Performance}_X = \frac{1}{\text{Execution time}_X} \]

- “Machine X is n times faster than Y”
  \[ n = \frac{\text{Performance}_X}{\text{Performance}_Y} = \frac{\text{Execution time}_Y}{\text{Execution time}_X} \]

- Problem:
  Execution time: Machine A: 12 seconds, B: 20 seconds
  - A/B = .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

- Need a precise definition
  \[ \Rightarrow \text{“X is n times faster than Y”, } n > 1 \]
  - A is 1.67 times faster than B
Clock Cycles

- Operation of conventional computer is controlled by Clock “ticks”

- Clock rate (frequency) = cycles per second (Hertz)
  
  MHz = Million cycles per second

  GHz = Giga (Billion) cycles per second

- Cycle time = time between ticks = seconds per cycle
  
  A 2 GHz clock => Cycle time = 1/(2x10^9) s = 0.5 nanoseconds

- Instead of reporting execution time in seconds, we often use clock cycles

  \[
  \frac{\text{seconds}}{\text{program}} = \frac{\text{cycles}}{\text{program}} \times \frac{\text{seconds}}{\text{cycle}}
  \]
How many cycles are required for a program?

- Assumption: \# of cycles = \# of instructions

- Assumption is incorrect
  - Typically, different instructions take different number of clock cycles
    - Integer Multiply instructions are multi-cycle
    - Floating point instructions are multi-cycle
Now that we understand cycles

- A given program will require
  - some number of instructions (machine instructions)
  - some number of cycles
  - some number of seconds

- We have a vocabulary that relates these quantities:
  - cycle time (seconds per cycle)
  - clock rate (cycles per second)
  - CPI (cycles per instruction)
Now that we understand cycles

- A given program will require
  - some number of instructions (machine instructions)
  - some number of cycles
  - some number of seconds

- We have a vocabulary that relates these quantities:
  - cycle time (seconds per cycle)
  - clock rate (cycles per second)
  - CPI (cycles per instruction)
CPI Example 2.1

- Suppose we have two implementations of the same instruction set architecture (ISA).
  For some program,
  - Machine A has a clock cycle time of 10 ns. and a CPI of 2.0
  - Machine B has a clock cycle time of 20 ns. and a CPI of 1.2

Which machine is faster for this program, and by how much?
Compiler Optimization: Example 2.2

- A compiler designer is trying to decide between two code sequences for a particular machine with 3 instruction classes.
  - CPI: Class A = 1, Class B = 2, and Class C = 3
  - Code sequence X has 5 instructions:
    - 2 of A, 1 of B, and 2 of C
  - Code sequence Y has 6 instructions:
    - 4 of A, 1 of B, and 1 of C.
- Which sequence will be faster? How much?
Execution time: Contributing Factors

\[
\text{CPU time} = \frac{\text{Seconds}}{\text{Program}} \times \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}
\]

<table>
<thead>
<tr>
<th>Instruction Count</th>
<th>CPI</th>
<th>Clock Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Compiler</td>
<td>X</td>
<td>(X)</td>
</tr>
<tr>
<td>ISA</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Organization</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Technology</td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

- Improve performance => reduce execution time
  - Reduce instruction count (Programmer, Compiler)
  - Reduce cycles per instruction (ISA, Machine designer)
  - Reduce clock cycle time (Hardware designer, Process engineer)
Performance

- Performance is determined by program execution time
- Do any one of the following variables equal performance?
  - # of cycles to execute program?
  - # of instructions in program?
  - # of cycles per second?
  - average # of cycles per instruction?
  - average # of instructions per second?

- Common pitfall: thinking one of the variables is indicative of performance when it really isn’t.
## Performance Variation

The CPU execution time can be calculated as follows:

\[
\text{CPU Execution Time} = \text{Instruction Count} \times \text{CPI} \times \text{Clock Cycle Time}
\]

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

- Amdahl’s law sets limit on how much improvement can be made.
Amdahl’s Law: Example 2.3

- Example: A program runs in 100 seconds on a machine, with multiply responsible for 80 seconds of this time. How much do we have to improve the speed of multiplication if we want the program to run 4 times faster?

- Is it possible to get the program to run 5 times faster?
Each metric has a place and a purpose, and each can be misused.
Benchmarks

- Performance best determined by running a real application
  - Use programs typical of expected workload
  - Or, typical of expected class of applications
    e.g., compilers/editors, scientific applications, graphics, etc.

- Small benchmarks
  - nice for architects and designers
  - easy to standardize
  - can be abused

- SPEC (System Performance Evaluation Cooperative)
  - companies have agreed on a set of real program and inputs
  - can still be abused (Intel’s “other” bug)
  - valuable indicator of performance (and compiler technology)
## SPEC CPU 2000 Benchmarks

<table>
<thead>
<tr>
<th>Integer Benchmarks</th>
<th>FP Benchmarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>gzip</td>
<td>Compression</td>
</tr>
<tr>
<td>vpr</td>
<td>FPGA Circuit Placement &amp; Routing</td>
</tr>
<tr>
<td>gcc</td>
<td>The GNU C Compiler</td>
</tr>
<tr>
<td>mcf</td>
<td>Combinatorial Optimization</td>
</tr>
<tr>
<td>crafty</td>
<td>Chess Program</td>
</tr>
<tr>
<td>Parser</td>
<td>Word Processing Program</td>
</tr>
<tr>
<td>eon</td>
<td>Computer Visualization</td>
</tr>
<tr>
<td>perlbench</td>
<td>Perl Application</td>
</tr>
<tr>
<td>gap</td>
<td>Group Theory, Interpreter</td>
</tr>
<tr>
<td>vortex</td>
<td>Object Oriented Database</td>
</tr>
<tr>
<td>Twolf</td>
<td>Place and Route Simulator</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
SPEC Ratings Vs Clock Speed

See www.spec.org
MIPS, MFLOPS etc.

- MIPS - million instructions per second
  \[
  \text{MIPS} = \frac{\text{number of instructions executed in program}}{\text{execution time in seconds} \times 10^6} = \frac{\text{Clock rate}}{\text{CPI} \times 10^6}
  \]

- MFLOPS - million floating point operations per second
  \[
  \text{MFLOPS} = \frac{\text{number of floating point operations executed in program}}{\text{execution time in seconds} \times 10^6}
  \]

- Hard to relate MIPS/MFLOPS with execution time
  \[
  \text{CPU Execution Time} = \text{Instruction Count} \times \frac{\text{CPI}}{\text{Clock Cycle Time}}
  \]

- Highly program-dependent metrics
- Deceptive (See pages 268 - 269 in the text)
  - MIPS would be higher for a program using simple instructions (e.g. a loop of NOPs!)
### RISC Processor: Example 2.4

<table>
<thead>
<tr>
<th>Base Machine (Reg / Reg)</th>
<th>Op</th>
<th>Freq</th>
<th>Cycles</th>
<th>CPI(i)</th>
<th>% Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU</td>
<td>50%</td>
<td>1</td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Load</td>
<td>20%</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Store</td>
<td>10%</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Branch</td>
<td>20%</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- What is average CPI?
- What percentage of time is spent in each instruction class?
- How much faster would the machine be if a better data cache reduced the average load time to 2 cycles?

- How does this compare with using branch prediction to shave a cycle off the branch time?

- What if two ALU instructions could be executed at once?
Performance assessment is tricky

- Performance is specific to particular program(s)
  - Total execution time is a consistent summary of performance

- For a given architecture performance increases come from:
  - Increases in clock rate (without adverse CPI affects)
  - Improvements in processor organization that lower CPI
  - Compiler enhancements that lower CPI and/or instruction count

- Pitfall: expecting improvement in one aspect of a machine’s performance to affect the total performance

- You should not always believe everything you read! Read carefully!
Announcements

- **Reading Assignment**
  - Chapter 4. Assessing and Understanding Performance
    Sections 4.1 - 4.6

- **Homework 3: Due Mon., October 17th in class**
  4.1, 4.2, 4.6, 4.7, 4.8, 4.11, 4.12, 4.19, 4.22, 4.45

  **Additional Problem for Chapter 2:** Write a program in C or C++ to declare and initialize an array of two “ints.” Print a word at a non-word aligned address starting in the first element of the array. Run the program on a PC and on a SPARC Station.  *(Hint: use a pointer to access the word.)*

  A. What is the output on each computer?
  B. If the program behavior is different on the two computers give explanation.

- **Quiz**
  **When:** Mon., October 17th, First 10 minutes of the class
  **Topic:** Performance, Chapter 4  **Need:** Paper, pen