CS 61C:
Great Ideas in Computer Architecture

Performance
Iron Law, Amdahl’s Law

Instructors:
Nicholas Weaver & Vladimir Stojanovic

http://inst.eecs.berkeley.edu/~cs61c/
New-School Machine Structures
(It’s a bit more complicated!)

- **Parallel Requests**
  Assigned to computer
e.g., Search “Katz”

- **Parallel Threads**
  Assigned to core
e.g., Lookup, Ads

- **Parallel Instructions**
  >1 instruction @ one time
e.g., 5 pipelined instructions

- **Parallel Data**
  >1 data item @ one time
e.g., Add of 4 pairs of words

- **Hardware descriptions**
  All gates @ one time

- **Programming Languages**

---

Software

Hardware

**Harness Parallelism & Achieve High Performance**

How do we know?
What is Performance?

• *Latency* (or *response time* or *execution time*)
  – Time to complete one task

• *Bandwidth* (or *throughput*)
  – Tasks completed per unit time
    • If you have sufficient independent tasks, you can always throw more money at the problem: 
      Throughput/$ often a more important metric than just throughput
Cloud Performance: Why Application Latency Matters

- Key figure of merit: application responsiveness
  - Longer the delay, the fewer the user clicks, the less the user happiness, and the lower the revenue per user

<table>
<thead>
<tr>
<th>Server Delay (ms)</th>
<th>Increased time to next click (ms)</th>
<th>Queries/ user</th>
<th>Any clicks/ user</th>
<th>User satisfaction</th>
<th>Revenue/ User</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>200</td>
<td>500</td>
<td>--</td>
<td>-0.3%</td>
<td>-0.4%</td>
<td>--</td>
</tr>
<tr>
<td>500</td>
<td>1200</td>
<td>--</td>
<td>-1.0%</td>
<td>-0.9%</td>
<td>-1.2%</td>
</tr>
<tr>
<td>1000</td>
<td>1900</td>
<td>-0.7%</td>
<td>-1.9%</td>
<td>-1.6%</td>
<td>-2.8%</td>
</tr>
<tr>
<td>2000</td>
<td>3100</td>
<td>-1.8%</td>
<td>-4.4%</td>
<td>-3.8%</td>
<td>-4.3%</td>
</tr>
</tbody>
</table>

Figure 6.10 Negative impact of delays at Bing search server on user behavior [Brutlag and Schurman 2009].
Defining CPU Performance

• What does it mean to say X is faster than Y?
• Ferrari vs. School Bus?
  • 2013 Ferrari 599 GTB
    – 2 passengers, quarter mile in 10 secs
  • 2013 Type D school bus
    – 50 passengers, quarter mile in 20 secs

• Response Time (Latency): e.g., time to travel ¼ mile
• Throughput (Bandwidth): e.g., passenger-mi in 1 hour
Defining Relative CPU Performance

- Performance$_X$ = $1/$Program Execution Time$_X$
- Performance$_X$ > Performance$_Y$ => $1/$Execution Time$_X$ > $1/$Execution Time$_Y$ => Execution Time$_Y$ > Execution Time$_X$
- Computer X is N times faster than Computer Y
  Performance$_X$/Performance$_Y$ = N or
  Execution Time$_Y$/Execution Time$_X$ = N

- Bus to Ferrari performance:
  - Program: Transfer 1000 passengers for 1 mile
  - Bus: 3,200 sec, Ferrari: 40,000 sec
Measuring CPU Performance

• Computers use a clock to determine when events take place within hardware

  • Clock cycles: discrete time intervals
    – aka clocks, cycles, clock periods, clock ticks

  • Clock rate or clock frequency: clock cycles per second (inverse of clock cycle time)

  • 3 GigaHertz clock rate

    => clock cycle time = 1/(3x10⁹) seconds
    clock cycle time = 333 picoseconds (ps)
CPU Performance Factors

• To distinguish between processor time and I/O, *CPU time* is time spent in processor

• CPU Time/Program
  \[ = \text{Clock Cycles/Program} \times \text{Clock Cycle Time} \]

• Or
  \[ \text{CPU Time/Program} = \frac{\text{Clock Cycles/Program}}{\text{Clock Rate}} \]
Iron Law of Performance
by Emer and Clark

• A program executes instructions

• CPU Time/Program
  = Clock Cycles/Program × Clock Cycle Time
  = Instructions/Program
    × Average Clock Cycles/Instruction
    × Clock Cycle Time

• 1\textsuperscript{st} term called \textit{Instruction Count}

• 2\textsuperscript{nd} term abbreviated \textit{CPI} for average \textit{Clock Cycles Per Instruction}

• 3\textsuperscript{rd} term is \textit{1 / Clock rate}
Restating Performance Equation

- Time = \frac{\text{Seconds}}{\text{Program}} \times \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock Cycle}}
What Affects Each Component? A) Instruction Count, B) CPI, C) Clock Rate

<table>
<thead>
<tr>
<th></th>
<th>Affects What? (click in letter of component not affected)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm</td>
<td></td>
</tr>
<tr>
<td>Programming Language</td>
<td></td>
</tr>
<tr>
<td>Compiler</td>
<td></td>
</tr>
<tr>
<td>Instruction Set Architecture</td>
<td></td>
</tr>
</tbody>
</table>
### What Affects Each Component?

#### Instruction Count, CPI, Clock Rate

<table>
<thead>
<tr>
<th>Component</th>
<th>Affects What?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm</td>
<td>Instruction Count, CPI</td>
</tr>
<tr>
<td>Programming Language</td>
<td>Instruction Count, CPI</td>
</tr>
<tr>
<td>Compiler</td>
<td>Instruction Count, CPI</td>
</tr>
<tr>
<td>Instruction Set Architecture</td>
<td>Instruction Count, Clock Rate, CPI</td>
</tr>
</tbody>
</table>
Clickers

<table>
<thead>
<tr>
<th>Computer</th>
<th>Clock frequency</th>
<th>Clock cycles per instruction</th>
<th>#instructions per program</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1GHz</td>
<td>2</td>
<td>1000</td>
</tr>
<tr>
<td>B</td>
<td>2GHz</td>
<td>5</td>
<td>800</td>
</tr>
<tr>
<td>C</td>
<td>500MHz</td>
<td>1.25</td>
<td>400</td>
</tr>
<tr>
<td>D</td>
<td>5GHz</td>
<td>10</td>
<td>2000</td>
</tr>
</tbody>
</table>

- Which computer has the highest performance for a given program?
Workload and Benchmark

• **Workload**: Set of programs run on a computer
  – Actual collection of applications run or made from real programs to approximate such a mix
  – Specifies programs, inputs, and relative frequencies

• **Benchmark**: Program selected for use in comparing computer performance
  – Benchmarks form a workload
  – Usually standardized so that many use them
SPEC
(System Performance Evaluation Cooperative)

• Computer Vendor cooperative for benchmarks, started in 1989
• SPECCPU2006
  – 12 Integer Programs
  – 17 Floating-Point Programs
• Often turn into number where bigger is faster
• SPECratio: reference execution time on old reference computer divide by execution time on new computer to get an effective speed-up
## SPECINT2006 on AMD Barcelona

<table>
<thead>
<tr>
<th>Description</th>
<th>Instruction Count (B)</th>
<th>CPI</th>
<th>Clock cycle time (ps)</th>
<th>Execution Time (s)</th>
<th>Reference Time (s)</th>
<th>SPEC-ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interpreted string processing</td>
<td>2,118</td>
<td>0.75</td>
<td>400</td>
<td>637</td>
<td>9,770</td>
<td>15.3</td>
</tr>
<tr>
<td>Block-sorting compression</td>
<td>2,389</td>
<td>0.85</td>
<td>400</td>
<td>817</td>
<td>9,650</td>
<td>11.8</td>
</tr>
<tr>
<td>GNU C compiler</td>
<td>1,050</td>
<td>1.72</td>
<td>400</td>
<td>724</td>
<td>8,050</td>
<td>11.1</td>
</tr>
<tr>
<td>Combinatorial optimization</td>
<td>336</td>
<td>10.0</td>
<td>400</td>
<td>1,345</td>
<td>9,120</td>
<td>6.8</td>
</tr>
<tr>
<td>Go game</td>
<td>1,658</td>
<td>1.09</td>
<td>400</td>
<td>721</td>
<td>10,490</td>
<td>14.6</td>
</tr>
<tr>
<td>Search gene sequence</td>
<td>2,783</td>
<td>0.80</td>
<td>400</td>
<td>890</td>
<td>9,330</td>
<td>10.5</td>
</tr>
<tr>
<td>Chess game</td>
<td>2,176</td>
<td>0.96</td>
<td>400</td>
<td>837</td>
<td>12,100</td>
<td>14.5</td>
</tr>
<tr>
<td>Quantum computer simulation</td>
<td>1,623</td>
<td>1.61</td>
<td>400</td>
<td>1,047</td>
<td>20,720</td>
<td>19.8</td>
</tr>
<tr>
<td>Video compression</td>
<td>3,102</td>
<td>0.80</td>
<td>400</td>
<td>993</td>
<td>22,130</td>
<td>22.3</td>
</tr>
<tr>
<td>Discrete event simulation library</td>
<td>587</td>
<td>2.94</td>
<td>400</td>
<td>690</td>
<td>6,250</td>
<td>9.1</td>
</tr>
<tr>
<td>Games/path finding</td>
<td>1,082</td>
<td>1.79</td>
<td>400</td>
<td>773</td>
<td>7,020</td>
<td>9.1</td>
</tr>
<tr>
<td>XML parsing</td>
<td>1,058</td>
<td>2.70</td>
<td>400</td>
<td>1,143</td>
<td>6,900</td>
<td>6.0</td>
</tr>
</tbody>
</table>
Summarizing Performance ... 

<table>
<thead>
<tr>
<th>System</th>
<th>Rate (Task 1)</th>
<th>Rate (Task 2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>B</td>
<td>20</td>
<td>10</td>
</tr>
</tbody>
</table>

Clickers: Which system is faster?

A: System A
B: System B
C: Same performance
D: Unanswerable question!
... Depends Who’s Selling

<table>
<thead>
<tr>
<th>System</th>
<th>Rate (Task 1)</th>
<th>Rate (Task 2)</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>10</td>
<td>20</td>
<td>15</td>
</tr>
<tr>
<td>B</td>
<td>20</td>
<td>10</td>
<td>15</td>
</tr>
</tbody>
</table>

Average throughput

<table>
<thead>
<tr>
<th>System</th>
<th>Rate (Task 1)</th>
<th>Rate (Task 2)</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0.50</td>
<td>2.00</td>
<td>1.25</td>
</tr>
<tr>
<td>B</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
</tbody>
</table>

Throughput relative to B

<table>
<thead>
<tr>
<th>System</th>
<th>Rate (Task 1)</th>
<th>Rate (Task 2)</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>B</td>
<td>2.00</td>
<td>0.50</td>
<td>1.25</td>
</tr>
</tbody>
</table>

Throughput relative to A
Summarizing SPEC Performance

• Varies from 6x to 22x faster than reference computer

• Geometric mean of ratios:
  N-th root of product of N ratios
  – Geometric Mean gives same relative answer no matter what computer is used as reference

• Geometric Mean for Barcelona is 11.7
Administrivia

• Midterm 2 in the evening next Monday
• Project 2.1 grades in
Big Idea: Amdahl’s (Heartbreaking) Law

• Speedup due to enhancement E is

\[
\text{Speedup w/ E} = \frac{\text{Exec time w/o E}}{\text{Exec time w/ E}}
\]

• Suppose that enhancement E accelerates a fraction F (F < 1) of the task by a factor S (S > 1) and the remainder of the task is unaffected

\[
\text{Execution Time w/ E} = \text{Execution Time w/o E} \times \left[ (1-F) + \frac{F}{S} \right]
\]

\[
\text{Speedup w/ E} = \frac{1}{[ (1-F) + \frac{F}{S} ]}
\]
Big Idea: Amdahl’s Law

Speedup = \frac{1}{(1 - F) + \frac{F}{S}}

Example: the execution time of half of the program can be accelerated by a factor of 2. What is the program speed-up overall?

\[
\frac{1}{0.5 + 0.5} = \frac{1}{0.5 + 0.25} = 1.33
\]
Example #1: Amdahl’s Law

\[
\text{Speedup w/ } E = \frac{1}{(1-F) + \frac{F}{S}}
\]

• Consider an enhancement which runs 20 times faster but which is only usable 25% of the time
  Speedup w/ E = \( \frac{1}{(0.75 + 0.25/20)} = 1.31 \)

• What if its usable only 15% of the time?
  Speedup w/ E = \( \frac{1}{(0.85 + 0.15/20)} = 1.17 \)

• Amdahl’s Law tells us that to achieve linear speedup with 100 processors, none of the original computation can be scalar!

• To get a speedup of 90 from 100 processors, the percentage of the original program that could be scalar would have to be 0.1% or less
  Speedup w/ E = \( \frac{1}{(0.001 + 0.999/100)} = 90.99 \)
If the portion of the program that can be parallelized is small, then the speedup is limited. The non-parallel portion limits the performance.
Strong and Weak Scaling

- To get good speedup on a parallel processor while keeping the problem size fixed is harder than getting good speedup by increasing the size of the problem.
  - **Strong scaling**: when speedup can be achieved on a parallel processor without increasing the size of the problem
  - **Weak scaling**: when speedup is achieved on a parallel processor by increasing the size of the problem proportionally to the increase in the number of processors

- **Load balancing** is another important factor: every processor doing same amount of work
  - Just one unit with twice the load of others cuts speedup almost in half
Clickers/Peer Instruction

Suppose a program spends 80% of its time in a square root routine. How much must you speed up square root to make the program run 5 times faster?

\[
\text{Speedup w/ E} = \frac{1}{(1-F) + F/S}
\]

A: 5
B: 16
C: 20
D: 100
E: None of the above
And In Conclusion, ...

- Time (seconds/program) is measure of performance

\[
\text{Time} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock Cycle}}
\]

- Floating-point representations hold approximations of real numbers in a finite number of bits