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
- Parallel Hardware descriptions
  All gates @ one time
- Parallel Programming Languages

Review

- Five main components of a computer
  – Control, Datapath, Memory, Input, Output
- Five stages of MIPS instruction execution
  – Fetch instruction
  – Decode/read registers
  – ALU (including address add)
  – Memory access (for loads and stores)
  – Writeback (if needed)

What is Performance?

- Latency
  (or response time or execution time)
  – Time to complete one task
- Bandwidth
  (or throughput)
  – Tasks completed per unit time

Cloud Performance:
Why Application Latency Matters

<table>
<thead>
<tr>
<th>Server Delay (ms)</th>
<th>Increased time to next click (ms)</th>
<th>Quizzes/ 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>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>500</td>
<td>1200</td>
<td>-1.0%</td>
<td>-0.9%</td>
<td>-1.2%</td>
<td>-2.9%</td>
</tr>
<tr>
<td>1000</td>
<td>1900</td>
<td>-1.8%</td>
<td>-1.9%</td>
<td>-1.8%</td>
<td>-2.8%</td>
</tr>
<tr>
<td>2000</td>
<td>3100</td>
<td>-2.1%</td>
<td>-2.3%</td>
<td>-3.8%</td>
<td>-4.1%</td>
</tr>
</tbody>
</table>

Figure 6.10 Negative impact of delays at Bing search server on user behavior [Bray and Schurmann 2009].

- 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

Google Instant Search
“Instant Efficiency”

Typical search takes 24 seconds, Google’s search algorithm is only 300 ms of this
“It’s not search ‘as you type’, but ‘search before you type’!”
“We can predict what you are likely to type and give you those results in real time”
Defining CPU Performance

• What does it mean to say X is faster than Y?
• Ferrari vs. School Bus?
  2009 Ferrari 599 GTB
  – 2 passengers, 11.1 secs in quarter mile
  2009 Type D school bus
  – 54 passengers, quarter mile time?
  http://www.youtube.com/watch?v=KwyCoQulJUNA
• 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 (=)\n  1/Execution Time_x > 1/Execution Time_y (=)\n  Execution Time_y > Execution Time_x
• Computer X is N times faster than Computer Y\n  Performance_x / Performance_y = N\n  or Execution Time_y / Execution Time_x = N
• Bus is to Ferrari as 12 is to 11.1: Ferrari is 1.08 times faster than the bus!

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/(3 \times 10^9) 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 = Clock Cycles/Program
  \times Clock Cycle Time
• Or
  CPU Time/Program = Clock Cycles/Program \div Clock Rate
• But a program executes instructions
• CPU Time/Program = Clock Cycles/Program \times Clock Cycle Time
  = Instructions/Program \times Average Clock Cycles/Instruction
  \times Clock Cycle Time
• 1st term called Instruction Count
• 2nd term abbreviated CPI for average Clock Cycles Per Instruction
• 3rd term is 1 / Clock rate

Restating Performance Equation

• Time = \frac{Seconds}{Program} \times \frac{Instructions}{Program} \times \frac{1}{Clock Cycles \times Seconds}{Instruction \times Clock Cycle}

This is the “Iron Law” of processor performance
What Affects Each Component?
Instruction Count, CPI, Clock Rate

<table>
<thead>
<tr>
<th>Hardware or software component?</th>
<th>Affects What?</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?

Instruc\textcolor{red}{\text{tion}} Count, CPI, Clock Rate

Hardware or software component?

Algorithm

Programming Language

Compiler

Instruction Set Architecture

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 both programs and relative frequencies
• **Benchmark**: Program selected for use in comparing computer performance
  – Benchmarks form a workload
  – Usually standardized so that many use them

SPECT\textcolor{red}{\text{EC}} (System Performance Evaluation Cooperative)

• Computer Vendor cooperative for benchmarks, started in 1989
• SPECT\textcolor{red}{\text{EC}CPU2006}
  – 12 Integer Programs
  – 17 Floating-Point Programs
• Often turn into number where bigger is faster
  • **SPEC\textcolor{red}{\text{ratio}}**: reference execution time on old reference computer divide by execution time on new computer to get an effective speed-up

SPEC\textcolor{red}{\text{INT2006}} 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\textcolor{red}{\text{ratio}}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interpreted string processing</td>
<td>2,118 0.75</td>
<td>400</td>
<td>637</td>
<td>9,770</td>
<td>15.3</td>
<td></td>
</tr>
<tr>
<td>Block-sorting compression</td>
<td>2,389 0.85</td>
<td>400</td>
<td>817</td>
<td>9,650</td>
<td>11.8</td>
<td></td>
</tr>
<tr>
<td>GNU C compiler</td>
<td>1,050 1.72</td>
<td>400</td>
<td>724</td>
<td>8,050</td>
<td>11.1</td>
<td></td>
</tr>
<tr>
<td>Combinatorial optimization</td>
<td>336 10.0</td>
<td>400</td>
<td>1,345</td>
<td>9,120</td>
<td>6.8</td>
<td></td>
</tr>
<tr>
<td>Go game</td>
<td>1,658 1.09</td>
<td>400</td>
<td>721</td>
<td>10,490</td>
<td>14.6</td>
<td></td>
</tr>
<tr>
<td>Search gene sequence</td>
<td>2,783 0.80</td>
<td>400</td>
<td>890</td>
<td>9,330</td>
<td>10.5</td>
<td></td>
</tr>
<tr>
<td>Chess game</td>
<td>2,176 0.96</td>
<td>400</td>
<td>837</td>
<td>12,100</td>
<td>14.5</td>
<td></td>
</tr>
<tr>
<td>Quantum computer simulation</td>
<td>1,623 1.61</td>
<td>400</td>
<td>1,047</td>
<td>20,720</td>
<td>19.8</td>
<td></td>
</tr>
<tr>
<td>Video compression</td>
<td>3,102 0.80</td>
<td>400</td>
<td>993</td>
<td>22,130</td>
<td>22.3</td>
<td></td>
</tr>
<tr>
<td>Discrete event simulation library</td>
<td>587 2.94</td>
<td>400</td>
<td>690</td>
<td>6,250</td>
<td>9.1</td>
<td></td>
</tr>
<tr>
<td>Games/path finding</td>
<td>1,082 1.79</td>
<td>400</td>
<td>773</td>
<td>7,020</td>
<td>9.1</td>
<td></td>
</tr>
<tr>
<td>XML parsing</td>
<td>1,058 2.70</td>
<td>400</td>
<td>1,143</td>
<td>6,900</td>
<td>6.0</td>
<td></td>
</tr>
</tbody>
</table>

Summarizing Performance ...

Flashcard Quiz: Which system is faster?

System A

System B

Same performance

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:
  \[
  \sqrt[N]{\prod_{i=1}^{N} \text{Execution time ratio}_i}
  \]
  - Geometric Mean gives same relative answer no matter what computer is used as reference
- Geometric Mean for Barcelona is 11.7

Administrivia

Energy and Power

(\text{Energy} = \text{Power} \times \text{Time})

- Energy to complete operation (Joules)
  - Corresponds approximately to battery life
- Peak power dissipation (\text{Watts} = \text{Joules/s})
  - Affects heat (and cooling demands)
- IT equipment’s power is in the denominator of the Power Utilization Efficiency (PUE) equation, a WSC figure of merit

Peak Power vs. Lower Energy

\text{(Power} \times \text{Time} = \text{Energy})

- Which system has higher peak power?
- Which system has higher energy?

Energy “Iron Law”

\[
\text{Performance} = \frac{\text{Power} \times \text{Energy Efficiency}}{\text{(Tasks/Second)} \times \text{Joules/Second}} = \frac{\text{(Tasks/Joule)}}{\text{Power Utilization Efficiency (PUE) equation, a WSC figure of merit}}
\]
Figure 1. Average CPU utilization of more than 5,000 servers during a six-month period. Servers are rarely completely idle and seldom operate near their maximum utilization, instead operating most of the time at between 10 and 50 percent of their maximum capacity. It is surprisingly hard to achieve high levels of utilization of typical servers (and your home PC or laptop is even worse).

**SPECPower**

- Increasing importance of power and energy: create benchmark for performance and power
- Most servers in WSCs have average utilization between 10% & 50%, so measure power at medium as well as at high load
- Measure best performance and power, then step down request rate to measure power for every 10% reduction in performance

**Java server benchmark performance is operations per second (ssj_ops), so metric is ssj_ops/Watt**

<table>
<thead>
<tr>
<th>Target Load (%)</th>
<th>Performance (ssj_ops)</th>
<th>Avg. Power (Watts)</th>
</tr>
</thead>
<tbody>
<tr>
<td>100%</td>
<td>231,867</td>
<td>295</td>
</tr>
<tr>
<td>90%</td>
<td>211,282</td>
<td>286</td>
</tr>
<tr>
<td>80%</td>
<td>185,803</td>
<td>275</td>
</tr>
<tr>
<td>70%</td>
<td>163,427</td>
<td>265</td>
</tr>
<tr>
<td>60%</td>
<td>140,160</td>
<td>256</td>
</tr>
<tr>
<td>50%</td>
<td>118,324</td>
<td>246</td>
</tr>
<tr>
<td>40%</td>
<td>92,035</td>
<td>233</td>
</tr>
<tr>
<td>30%</td>
<td>70,500</td>
<td>222</td>
</tr>
<tr>
<td>20%</td>
<td>47,126</td>
<td>206</td>
</tr>
<tr>
<td>10%</td>
<td>23,066</td>
<td>180</td>
</tr>
<tr>
<td>0%</td>
<td>0</td>
<td>141</td>
</tr>
<tr>
<td>Sum</td>
<td>1,283,590</td>
<td>2,605</td>
</tr>
</tbody>
</table>

**Which is Better? (1 Red Machine vs. 5 Green Machines)**

- Five machines running at 10% utilization
  - Total Power =
- One machine running at 50% utilization
  - Total Power =

**Other Benchmark Attempts**

- Rather than run a collection of real programs and take their average (geometric mean), create a single program that matches the average behavior of a set of programs
- Called a **synthetic benchmark**
- First example called **Whetstone** in 1972 for floating-point-intensive programs in Fortran
- Second example called **Dhrystone** in 1985 for integer programs in Ada and C
  - Pun on Wet vs. Dry ("Whet" vs. "Dhry")

**Dhrystone Shortcomings**

- Dhrystone features unusual code that is not usually representative of real-life programs
- Dhrystone susceptible to compiler optimizations
- Dhrystone’s small code size means always fits in caches (<2KB), so not representative
- Yet still used in hand-held, embedded CPUs!
EE Times Articles

“Samsung and Intrinsity announced they have 1st silicon for Hummingbird, an ARM Cortex A8 that ... delivers more than 2,000 Dhrystone Mips while consuming 640 mW power” 7/24/09

Compiled Size of Dhystone 9/7/2010

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Embedded 32-bit</th>
<th>MIPS Embedded 32-bit</th>
<th>RISC Embedded Microblaze 40XT</th>
<th>ARM Cortex A8</th>
<th>ARM Cortex A9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Program size in bytes</td>
<td>8 BIT</td>
<td>16 BIT</td>
<td>32 BIT</td>
<td>40BIT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3196</td>
<td>920</td>
<td>5979</td>
<td>912</td>
<td>980</td>
</tr>
</tbody>
</table>

How to get RDTSC access in C?

```
static inline unsigned long long RDTSC(void)
{
    unsigned hi, lo;
    asm volatile("rdtsc":=a"(lo), =d"(hi));
    return ((unsigned long long)lo) | ((unsigned long long)hi) <<32 );
}
```

gcc Optimization Experiment

```
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BubbleSort.c</td>
<td>Dhrystone.c</td>
</tr>
<tr>
<td>No Opt</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-O1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-O2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-O3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Measuring Time

- UNIX time command measures in seconds
- Time Stamp Counter
  - 64-bit counter of clock cycles on Intel 80x86 instruction set computers
  - 80x86 instruction RDTSC (Read TSC) returns TSC in reg EDX (upper 32 bits) and EAX (lower 32 bits)
  - Can read, but can’t set
  - How long can measure?
  - Measures overall time, not just time for 1 program

Where Do You Spend the Time in Your Program?

- Profiling a program (e.g., using, gprof) shows where it spends its time by function, so you can determine which code consumes most of the execution time
- Usually a 90/10 rule: 10% of code is responsible for 90% of execution time
  - Or 80/20 rule, where 20% of code responsible for 80% of time

```
gprof
```

- Learn where program spent its time
- Learn functions called while it was executing
  - And which functions call other functions
- Three steps:
  - Compile & link program with profiling enabled
    - cc -pg x.c (in addition to other flags use)
  - Execute program to generate a profile data file
  - Run gprof to analyze the profile data
**gprof example**

<table>
<thead>
<tr>
<th>Cumulative</th>
<th>Self</th>
<th>Self</th>
<th>Total</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>(secs)</td>
<td>(secs)</td>
<td>(secs)</td>
<td>(call)</td>
<td></td>
</tr>
<tr>
<td>18.18</td>
<td>0.06</td>
<td>0.06</td>
<td>23480</td>
<td>0.00 find_char_unquote</td>
</tr>
<tr>
<td>12.12</td>
<td>0.10</td>
<td>0.04</td>
<td>120</td>
<td>0.33 pattern_search</td>
</tr>
<tr>
<td>9.09</td>
<td>0.13</td>
<td>0.03</td>
<td>5120</td>
<td>0.01 collapse_continuations</td>
</tr>
<tr>
<td>9.09</td>
<td>0.16</td>
<td>0.03</td>
<td>148</td>
<td>0.20 update_file_1</td>
</tr>
<tr>
<td>9.09</td>
<td>0.19</td>
<td>0.03</td>
<td>37</td>
<td>0.81 eval</td>
</tr>
<tr>
<td>6.06</td>
<td>0.21</td>
<td>0.02</td>
<td>12484</td>
<td>0.00 file_hash_1</td>
</tr>
<tr>
<td>6.06</td>
<td>0.23</td>
<td>0.02</td>
<td>6596</td>
<td>0.00 get_next_mword</td>
</tr>
<tr>
<td>3.03</td>
<td>0.24</td>
<td>0.01</td>
<td>29981</td>
<td>0.00 hash_find_slot</td>
</tr>
<tr>
<td>3.03</td>
<td>0.25</td>
<td>0.01</td>
<td>14769</td>
<td>0.00 next_token</td>
</tr>
<tr>
<td>3.03</td>
<td>0.26</td>
<td>0.01</td>
<td>5800</td>
<td>0.00 variable_expand_string</td>
</tr>
</tbody>
</table>

See [http://knuxgazette.net/100/vinayak.html](http://knuxgazette.net/100/vinayak.html)

---

**Cautionary Tale**

- “More computing sins are committed in the name of efficiency (without necessarily achieving it) than for any other single reason - including blind stupidity”
  -- William A. Wulf
- “We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil”
  -- Donald E. Knuth

---

**And In Conclusion, ...**

- Time (seconds/program) is measure of performance
  \[
  \text{Time} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock Cycle}}
  \]
- Benchmarks stand in for real workloads to as standardized measure of relative performance
- Power of increasing concern, and being added to benchmarks
- Time measurement via clock cycles, machine specific
- Profiling tools as way to see where spending time in your program
- Don’t optimize prematurely!