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

Review

- Everything is a (binary) number in a computer
  - Instructions and data; stored program concept
- Assemblers can enhance machine instruction set to help assembly-language programmer
- Translate from text that easy for programmers to understand into code that machine executes efficiently:
  - Compilers, Assemblers
- Linkers allow separate translation of modules
- Interpreters for debugging, but slow execution
- Hybrid (Java): Compiler + Interpreter to try to get best of both
- Compiler Optimization to relieve programmer

What is Performance?

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

Running Systems to 100% Utilization

- Implication of the graph at the right?
- Can you explain why this happens?

Agenda

- Defining Performance
- Administrivia
- Workloads and Benchmarks
- Technology Break
- Measuring Performance
- Summary
The Iron Law of Queues
(aka Little’s Law)

\[ L = \lambda \cdot W \]

Average number of customers in system (L) = average interarrival rate (\( \lambda \)) x average service time (W)

Cloud Performance:
Why Application Latency Matters

<table>
<thead>
<tr>
<th>Server Delay (ms)</th>
<th>Increased time to next clock (ms)</th>
<th>Queues/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>-0.6%</td>
<td>--</td>
</tr>
<tr>
<td>500</td>
<td>1200</td>
<td>-1.0%</td>
<td>-0.9%</td>
<td>-1.2%</td>
<td>--</td>
</tr>
<tr>
<td>1000</td>
<td>1900</td>
<td>-6.7%</td>
<td>-1.5%</td>
<td>-1.8%</td>
<td>-2.8%</td>
</tr>
<tr>
<td>2000</td>
<td>3100</td>
<td>-13.8%</td>
<td>-4.4%</td>
<td>-3.8%</td>
<td>-4.3%</td>
</tr>
</tbody>
</table>

Figure 6.10: Negative impact of delays on player search server on user behavior (Breidig and Schumann 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 D engine bus
    - 54 passengers, quarter mile time?
    http://www.youtube.com/watch?v=KwyCoQubUNA
  • 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\(_Y\)
• Computer X is N times faster than Computer Y
  Performance\(_X\) / Performance\(_Y\) = 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/(3x10\(^9\)) 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 × Clock Cycle Time
- Or
  - CPU Time/Program
  - = Clock Cycles/Program + Clock Rate

CPU Performance Factors

- But a program executes instructions
- CPU Time/Program
  - = Clock Cycles/Program × Clock Cycle Time
  - = Instructions/Program × Average Clock Cycles/Instruction × 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 = \( \frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock Cycle}} \)

What Affects Each Component?

<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>

Computer A clock cycle time 250 ps, CPI_A = 2
Computer B clock cycle time 500 ps, CPI_B = 1.2
Assume A and B have same instruction set
Which statement is true?
- Computer A is ≈1.2 times faster than B
- Computer A is ≈4.0 times faster than B
- Computer B is ≈1.7 times faster than A
- **Computer B is ≈3.4 times faster than A**

Administrivia

- Lab #5 posted
- Project #2.1 Due Sunday @ 11:59:59
- HW #4 Due Sunday @ 11:59:59
- Midterm in less than three weeks:
  - No discussion during exam week
  - TA Review: Su, Mar 4, starting 2 PM, 2050 VLSB
  - Exam: Tu, Mar 6, 6:40-9:40 PM, 2050 VLSB (room change)
  - Small number of special consideration cases, due to class conflicts, etc.—contact me
Getting to Know Profs

- Ride with 2 sons in MS Charity Bike Ride every September since 2002
  - “Waves to Wine”
  - 150 miles over 2 days from SF to Sonoma
- Team: “Berkeley Anti-M5 Crew”
  - If want to join team, let me know
  - Always a Top 10 fundraising team despite small size

  - Can offer fund raising advice: order of sending, when to send during week, who to send to, ...

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

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

<table>
<thead>
<tr>
<th>SPECINT2006 on AMD Barcelona</th>
<th>System</th>
<th>Rate (Task 1)</th>
<th>Rate (Task 2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interpreted string processing</td>
<td>A</td>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>Block-sorting compression</td>
<td>B</td>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>GNU C compiler</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Combinatorial optimization</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Go game</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Search gene sequence</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Chess game</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Quantum computer simulation</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Video compression</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Discrete event simulation library</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Games/path finding</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>XSLT parsing</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Summarizing Performance ...

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

*Which system is faster?*
... 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>0.50</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>1.00</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: \( \sqrt[N]{\prod_{i=1}^{N} \text{Execution time ratio}_i} \)
  - \( \text{Geometric Mean gives same relative answer no matter what computer is used as reference} \)
  - Geometric Mean for Barcelona is 11.7

Energy and Power

(Energy = Power x Time)

• Energy to complete operation (Joules)
  - Corresponds approximately to battery life

• Peak power dissipation (Watts = 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

(Power x Time = Energy)

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

Energy Proportional Computing

“The Case for Energy-Proportional Computing,”
Luiz André Barroso, Urs Hölzle,
IEEE Computer, December 2007

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

\[
\text{overall ssj_ops per Watt} = \frac{\sum_{i=8}^{50} \text{ssj_ops}_i}{\sqrt{\sum_{i=8}^{50} \text{power}_i}}
\]
**SPEC Power on Barcelona**

<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, so not representative
- Yet still used in hand held, embedded CPUs!

**EE Times Articles**

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

Compiled Size of Dhrystone 9/7/2010

| Architecture | Balanced (MIPS) | ARM (MIPS) | ARM (Gflops) | ARM (Gflops)
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Task</td>
<td>Load</td>
<td>Embedded</td>
<td>Load</td>
<td>Embedded</td>
</tr>
<tr>
<td></td>
<td>3.0 GHz</td>
<td>Microbench</td>
<td>2.6 GHz</td>
<td>Microbench</td>
</tr>
<tr>
<td></td>
<td>800 MHz</td>
<td>1024x400</td>
<td>800 MHz</td>
<td>1024x400</td>
</tr>
<tr>
<td>Program</td>
<td>3106</td>
<td>920</td>
<td>1974</td>
<td>912</td>
</tr>
<tr>
<td>Run time</td>
<td>8 BIT</td>
<td>16 BIT</td>
<td>16 BIT</td>
<td>32 BIT</td>
</tr>
<tr>
<td>Type*</td>
<td>8 BIT</td>
<td>16 BIT</td>
<td>16 BIT</td>
<td>32 BIT</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 RDTS (Read TSC) returns TSC in regs 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
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

<table>
<thead>
<tr>
<th></th>
<th>BubbleSort.c</th>
<th>Dhrystone.c</th>
</tr>
</thead>
<tbody>
<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>

gprof

- Learn where program spent its time
- Learn which functions called while it was executing
- Three steps:
  - Compile & link program with profiling enabled
    - cc -pg x.c
  - Execute program to generate a profile data file
  - Run gprof to analyze the profile data

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

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

See http://linuxgazette.net/130/vinayak.html
And In Conclusion, ...

- Time (seconds/program) is measure of performance
  \[ \text{Time (seconds/program)} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock Cycle}} \]

- 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!