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>Quantity</th>
<th>Any clicks/</th>
<th>User satisfaction</th>
<th>Revenue/User</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>200</td>
<td>500</td>
<td>-0.3%</td>
<td>-0.4%</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>--</td>
</tr>
<tr>
<td>1000</td>
<td>1900</td>
<td>0.7%</td>
<td>1.4%</td>
<td>1.6%</td>
<td>2.4%</td>
</tr>
<tr>
<td>2000</td>
<td>3100</td>
<td>1.4%</td>
<td>4.4%</td>
<td>3.4%</td>
<td>4.3%</td>
</tr>
</tbody>
</table>

Figure 6.10: Negative impact of delays at Bing search servers on user behavior [Bierling and Schuman 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.

CPU Iron Law!

Restating Performance Equation

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

Instruction Count, CPI, Clock Rate

Peer Instruction

<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>RED</td>
<td>1.0 GHz</td>
<td>2</td>
<td>1800</td>
</tr>
<tr>
<td>GREEN</td>
<td>2.0 GHz</td>
<td>5</td>
<td>800</td>
</tr>
<tr>
<td>ORANGE</td>
<td>0.5 GHz</td>
<td>1.25</td>
<td>400</td>
</tr>
<tr>
<td>YELLOW</td>
<td>5.0 GHz</td>
<td>10</td>
<td>2000</td>
</tr>
</tbody>
</table>

Which computer has the highest performance for a given program?

Which system is faster?

RED: System A
GREEN: System B
ORANGE: Same performance
YELLOW: Unanswerable question!

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 divided by execution time on new computer, reported as speed-up
- (SPEC 2017 just released)

SPECINT2006 on AMD Barcelona

<table>
<thead>
<tr>
<th>Description</th>
<th>Instruction Count ($)</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>Interpolated string stretching</td>
<td>2,118</td>
<td>0.75</td>
<td>400</td>
<td>631</td>
<td>9,770</td>
<td>15.3</td>
</tr>
<tr>
<td>Black-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>Convolutional 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>def 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>Class game</td>
<td>2,176</td>
<td>0.96</td>
<td>400</td>
<td>837</td>
<td>12,100</td>
<td>14.8</td>
</tr>
<tr>
<td>Quantum computer</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>Graph drawing</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 SPEC Performance

- Varies from 6x to 22x faster than reference computer
- Geometric mean of rat 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

Outline

- Defining Performance
- Floating-Point Standard
- And in Conclusion ...

Administrivia

- Midterm 2 in one week!
- Guerrilla Session tonight 7-9pm in Cory 203
- ONE double-sided cheat sheet
- Review session Saturday 10-12pm in Cory 203
- Homework 4 released!
- Caches and Floating Point
- Due after the midterm
- Still good cache practice!
- Proj2-Part2 released
- Due after the midterm, but good to do before!

Quick Number Review

- Computers deal with numbers
- What can we represent in N bits?
  - $2^N$ things, and no more! They could be...
    - Unsigned integers: $0$ to $2^N - 1$ (for $N=32$, $2^{32} - 1 = 4,294,967,295$)
    - Signed Integers (Two’s Complement) $-2^{(N-1)}$ to $2^{(N-1)} - 1$ (for $N=32$, $2^{31} = 2,147,483,648$)

Other Numbers

1. Very large numbers? (seconds/millennium) $\approx 31,556,926,000,000,000,000 (3.15692621 \times 10^{13})$
2. Very small numbers? (Bohr radius) $\approx 0.000000000052917710$m (5.29177 x 10^{-11})
3. Numbers with both integer & fractional parts? $\approx 1.5$
- First consider #3
- ...our solution will also help with #1 and #2

Goals for Floating Point

- Standard arithmetic for reals for all computers
- Keep as much precision as possible in formats
- Help programmer with errors in real arithmetic
  - $\pm \infty$, Not-A-Number (NaN), exponent overflow, exponent underflow
- Keep encoding that is somewhat compatible with two’s complement
  - E.g., 0 in Fl. Pt. is 0 in two’s complement
  - Make it possible to sort without needing to do floating-point comparison
Representation of Fractions

“Binary Point” like decimal point signifies boundary between integer and fractional parts:

Example 6-bit representation:
\[
\begin{array}{ccccccc}
XX & . & YYYY \\
2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 & 2^{-1} \\
10.1010 &=& 1x2^1 &+& 1x2^{-1} &+& 1x2^{-3} &=& 2.625_{ten}
\end{array}
\]

If we assume “fixed binary point”, range of 6-bit representations with this format: 0 to 3.9375 (almost 4)

10/24/17

Fractional Powers of 2

\[
\begin{array}{|c|c|c|c|c|c|c|}
\hline
i & 2^{-i} \\
\hline
0 & 1.0 \\
1 & 0.5 & 1/2 \\
2 & 0.25 & 1/4 \\
3 & 0.125 & 1/8 \\
4 & 0.0625 & 1/16 \\
5 & 0.03125 & 1/32 \\
6 & 0.015625 & 1/64 \\
7 & 0.0078125 & 1/128 \\
8 & 0.00390625 & 1/256 \\
9 & 0.001953125 & 1/512 \\
10 & 0.0009765625 & 1/1024 \\
11 & 0.00048828125 & 1/2048 \\
12 & 0.000244140625 & 1/4096 \\
13 & 0.0001220703125 & 1/8192 \\
14 & 0.00006103515625 & 1/16384 \\
15 & 0.000030517578125 & 1/32768 \\
\hline
\end{array}
\]

10/24/17

Scientific Notation (in Decimal)

- Mantissa
- Exponent
- Radix (base)

- Normalized form: no leadings 0s (exactly one digit to left of decimal point)
- Alternatives to representing 1/1,000,000,000
  - Normalized: \(1.0 \times 10^{-9}\)
  - Not normalized: \(0.1 \times 10^8, 10.0 \times 10^{-10}\)

10/24/17

Scientific Notation (in Binary)

- Mantissa
- Exponent
- “Binary point”
- Radix (base)

- Computer arithmetic that supports it is called floating point, because it represents numbers where the binary point is not fixed, as it is for integers
  - Declare such variable in C as float
  - double for double precision

10/24/17
Floating-Point Representation (1/4)

- 32-bit word has 2^{32} patterns, so must be approximation of real numbers ≥ 1.0, < 2
- IEEE 754 Floating-Point Standard:
  - 1 bit for sign (s) of floating point number
  - 8 bits for exponent (E)
  - 23 bits for fraction (F) (get 1 extra bit of precision if leading 1 is implicit)
  \((-1)^s \times (1 + F) \times 2^E\)
- Can represent from 2.0 x 10^{-38} to 2.0 x 10^{38}

Floating-Point Representation (2/4)

- Normal format: \(+1.xxx...x_{\text{two}} \times 2^{\text{YY - Y}_{\text{two}}}\)
- Multiple of Word Size (32 bits)

Floating-Point Representation (3/4)

- What if result too large?
  \((-2.0\times10^{38}, < 2.0\times10^{38})\)
  - Overflow: Exponent larger than represented in 8-bit Exponent field
- What if result too small?
  \((0.0 < 2.0\times10^{-38}, > 0.0 \times 2.0\times10^{-38})\)
  - Underflow: Negative exponent larger than represented in 8-bit Exponent field
- What would help reduce chances of overflow and/or underflow?

Floating-Point Representation (4/4)

- What about bigger or smaller numbers?
- IEEE 754 Floating-Point Standard:
  - Double Precision (64 bits)
  - 1 bit for sign (s) of floating-point number
  - 11 bits for exponent (E)
  - 52 bits for fraction (F)
  \((-1)^s \times (1 + F) \times 2^E\)
  - Can represent from 2.0 x 10^{-308} to 2.0 x 10^{308}

Which is Less?
(i.e., closer to -∞)

- 0 vs. 1 x 10^{-127}?
- 1 x 10^{-126} vs. 1 x 10^{-127}?
- -1 x 10^{-127} vs. 0?
- -1 x 10^{-126} vs. -1 x 10^{-127}?

Floating Point: Representing Very Small Numbers

- Zero: Bit pattern of all 0s is encoding for 0.000
  \(\Rightarrow 0\) in exponent should mean most negative exponent (want 0 to be next to smallest real)
  \(\Rightarrow 0\) can’t use two’s complement (1000 0000)
- Bias notation: Subtract bias from exponent
  \(-\) Single precision uses bias of 127; DP uses 1023
- 0 uses 0000 0000_{two} \(\Rightarrow 0-127 = -127; \approx\)
- NaN uses 1111 1111_{two} \(\Rightarrow 255-127 = +128\)
  - Smallest SP real can represent: 1.00...00 x 2^{-126}
  - Largest SP real can represent: 1.11...11 x 2^{127}
IEEE 754 Floating-Point Standard (1/3)

Single Precision (Double Precision similar):

<table>
<thead>
<tr>
<th>1 bit</th>
<th>8 bits</th>
<th>23 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Exponent</strong></td>
<td><strong>Significand</strong></td>
<td><strong>0</strong></td>
</tr>
</tbody>
</table>

- **Sign bit**: 1 means negative, 0 means positive
- **Significant in sign-magnitude format (not 2’s complement)**
  - To pack more bits, leading 1 implicit for normalized numbers
  - Always true: 0 < Significant < 1 (for normalized numbers)
- **Note**: 0 has no leading 1, so reserve exponent value 0 just for number 0

IEEE 754 Floating-Point Standard (2/3)

- IEEE 754 uses “biased exponent” representation
  - Designers wanted FP numbers to be used even if no FP hardware; e.g., sort records with FP numbers using integer compares
  - Wanted bigger (integer) exponent field to represent bigger numbers
  - 2’s complement poses a problem (because negative numbers look bigger)
  - Use just magnitude and offset by half the range

IEEE 754 Floating-Point Standard (3/3)

- **Summary (single precision)**:
  - Called **Biased Notation**, where bias is number subtracted to get final number
    - IEEE 754 uses bias of 127 for single precision
    - Subtract 127 from Exponent field to get actual exponent value
  - **Bias Notation (+127)**
    - How it is interpreted  
      - How it is encoded

<table>
<thead>
<tr>
<th>Decimal</th>
<th>sign</th>
<th>Exponent bias</th>
<th>Exponent</th>
<th>Biased Notation</th>
<th>Decimal Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Infinity</td>
<td>1</td>
<td>127</td>
<td>0</td>
<td>11111111</td>
<td>2^32</td>
</tr>
<tr>
<td>NaN</td>
<td>0</td>
<td>127</td>
<td>0</td>
<td>11111110</td>
<td>2^32</td>
</tr>
<tr>
<td>Get closer to zero</td>
<td></td>
<td>-127</td>
<td>0</td>
<td>11111111</td>
<td>1.0</td>
</tr>
<tr>
<td>+</td>
<td>1</td>
<td>127</td>
<td>0</td>
<td>11111111</td>
<td>2^0</td>
</tr>
<tr>
<td>Zero</td>
<td>0</td>
<td>-127</td>
<td>0</td>
<td>11111110</td>
<td>2^-127</td>
</tr>
<tr>
<td>For Denormal</td>
<td></td>
<td>-127</td>
<td>0</td>
<td>00000000</td>
<td>2^-127 - 2^-1023</td>
</tr>
</tbody>
</table>

Peer Instruction

- Guess this Floating Point number:
  - 1 10000000 10000000 00000000 00000000
  - **RED**: -1 x 2^128
  - **GREEN**: +1 x 2^128
  - **ORANGE**: -1 x 2^1
  - **YELLOW**: -1.5 x 2^1

More Floating Point

- What about 0?
  - Bit pattern all 0s means 0, so no implicit leading 1
- What if divide 1 by 0?
  - Can get infinity symbols \( \pm \infty \)
- What if do something stupid? (\( \frac{0}{0} \))
  - Can get special symbols NaN for “Not-a-Number”
- What if result is too big? (2x10^{308} x 2x10^{3})
  - Get overflow in exponent, alert programmer!
- What if result is too small? (2x10^{-308} x 2x10^{3})
  - Get underflow in exponent, alert programmer!
Representation for ± ∞

• In FP, divide by 0 should produce ± ∞, not overflow
• Why?
  – OK to do further computations with ± ∞
  – E.g., X/0 > Y may be a valid comparison
• IEEE 754 represents ± ∞
  – Most positive exponent reserved for ± ∞
  – Significand all zeroes

Representation for 0

• Represent 0?
  – Exponent all zeroes
  – Significand all zeroes
  – What about sign? Both cases valid
+0: 0 00000000 00000000000000000000000
−0: 1 00000000 00000000000000000000000

Special Numbers

• What have we defined so far? (Single Precision)

<table>
<thead>
<tr>
<th>Exponent</th>
<th>Significand</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1-254</td>
<td>nonzero</td>
<td>??</td>
</tr>
<tr>
<td>255</td>
<td>0</td>
<td>+/- fl. pt. #</td>
</tr>
<tr>
<td>255</td>
<td>nonzero</td>
<td>??</td>
</tr>
</tbody>
</table>

Representation for Not-a-Number

• What do I get if I calculate sqrt(-4.0) or 0/0?
  – If ± ∞ not an error, these shouldn’t be either
  – Called not a number (NaN)
  – Exponent = 255, Significand nonzero
• Why is this useful?
  – Hope NaNs help with debugging?
  – They contaminate: op(NaN, X) = NaN
  – Can use the significand to identify which!
  (e.g., quiet NaNs and signaling NaNs)

Representation for Denorms (1/2)

• Problem: There’s a gap among representable FP numbers around 0
  – Smallest representable positive number:
    a = 1.0… * 2^−126
  – Second smallest representable positive number:
    b = 1.000…1 * 2^−126
    = (1 + 0.00…1) * 2^−126
    = (1 + 2^−23) * 2^−126
    = 2^−126 + 2^−149
    a − 0 = 2^−126
    b − a = 2^−149

• Solution:
  – We still haven’t used Exponent = 0, Significand nonzero
  – Denormalized number: no (implied) leading 1, implicit exponent = -126
  – Smallest representable positive number:
    a = 2^−126 (i.e., 2^−126 * 2^−23)
  – Second-smallest representable positive number:
    b = 2^−149 (i.e., 2^−126 * 2^−22)

Representation for Denorms (2/2)

• Solution:
  – We still haven’t used Exponent = 0, Significand nonzero
  – Denormalized number: no (implied) leading 1, implicit exponent = -126
  – Smallest representable positive number:
    a = 2^−126 (i.e., 2^−126 * 2^−23)
  – Second-smallest representable positive number:
    b = 2^−149 (i.e., 2^−126 * 2^−22)
Special Numbers Summary

- Reserve exponents, significands:

<table>
<thead>
<tr>
<th>Exponent</th>
<th>Significand</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>nonzero</td>
<td>Denorm</td>
</tr>
<tr>
<td>1-254</td>
<td>anything</td>
<td>+/- fl. pt. number</td>
</tr>
<tr>
<td>255</td>
<td>0</td>
<td>+/- ∞</td>
</tr>
<tr>
<td>255</td>
<td>Nonzero</td>
<td>NaN</td>
</tr>
</tbody>
</table>

Saving Bits

- Many applications in machine learning, graphics, signal processing can make do with lower precision
  - IEEE “half-precision” or “FP16” uses 16 bits of storage
    - 1 sign bit
    - 5 exponent bits (exponent bias of 15)
    - 10 significand bits
  - Microsoft “BrainWave” FPGA neural net computer uses proprietary 8-bit and 9-bit floating-point formats

Outline

- Defining Performance
- Floating Point Standard
- And in Conclusion ...

And In Conclusion, ...

- Time (seconds/program) is measure of performance
  \[
  \text{Seconds} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock Cycle}}
  \]
- Floating-point representations hold approximations of real numbers in a finite number of bits
  - IEEE 754 Floating-Point Standard is most widely accepted attempt to standardize interpretation of such numbers (Every desktop or server computer sold since ~1997 follows these conventions)
  - Single Precision:
    \[
    \begin{array}{ccc|c|c|}
    \hline
    \hline
    \text{Exponent} & \text{Significand} & \text{0} & \text{1 bit} & \text{8 bits} & \text{23 bits} \\
    \hline
    \end{array}
    \]
