Performance and Floating-Point Arithmetic
Virtual Memory Paging As A Cache...

- How virtual memory works we will cover later...
- But for now, it's easy to model as a basic cache...
- Your program is given the illusion of as much RAM as it wants...
  - But only on the condition that it actually doesn't want it! :)
- Idea: Virtual memory can copy "pages" between RAM and disk
  - The main memory thus acts as a cache for the disk...
But What Are The Properties...

- **Associativity**: Effectively fully associative
- **Replacement policy**: Effectively full LRU
- **Block size**: 4kB or more
  - Some argue it should now be 64kB or 256kB these days
- **Hit time**...
  - Call it 0
- **Miss penalty**...
  - Latency to get a block from disk: 1ms or so for an SSD...
    Or put in clock terms, a 1 GHz clock -> 1,000,000 clock cycles!
  - Or if you have a spinning disk: 10ms or so...
    So **10,000,000 clock cycles**!
  - Upgrade your computer to an SSD if you haven't already!
So what are the implications?

• As long as you don't really use it, Virtual Memory is great...
  • Basically as long as your **working set** fits in physical memory, virtual memory is great at handling a little extra...

• But as soon as your working set exceeds physical memory, your performance goes to crap...
  • The system starts **thrashing**: Repeatedly needing to copy data to and from disk...
    • Similar to **thrashing the cache** when your working set exceeds cache capacity
  • If you have a spinning disk, it really starts sounding like the computer is suffering....
Outline

• Floating-Point Representation
• Defining and Measuring Performance
• And in Conclusion …
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

• Numbers with both integer & fractional parts?
  • ex: 1.5

• Very large numbers? (seconds/millennium)
  • ex: $3.155692610 \times 10^{10}$

• Very small numbers? (Bohr radius)
  ex: $5.2917710 \times 10^{-11}$ m

• First consider #1

• …our solution will also help with #2 and #3
Representation of Fractions

• Look at decimal (base 10) first:
• Decimal “point” signifies boundary between integer and fractional parts:

Example 6-digit representation:

25.2406\text{ten} = 2 \times 10^1 + 5 \times 10^0 + 2 \times 10^{-1} + 4 \times 10^{-2} + 6 \times 10^{-4}

If we assume “fixed decimal point”, range of 6-digit representations with this format: 0 to 99.9999. Not much range, but lots of “precision”
Binary Representation of Fractions

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

Example 6-bit representation:

\[ 10.1010_{\text{two}} = 1 \times 2^1 + 1 \times 2^{-1} + 1 \times 2^{-3} = 2.625_{\text{ten}} \]

If we assume “fixed binary point”, range of 6-bit representations with this format: 0 to 3.9375 (almost 4)
## Fractional Powers of 2

<table>
<thead>
<tr>
<th>$i$</th>
<th>$2^{-i}$</th>
<th>(base 2)</th>
<th>(base 10)</th>
<th>(fraction)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.0</td>
<td>1.0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.01</td>
<td>0.5</td>
<td>1/2</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0.001</td>
<td>0.25</td>
<td>1/4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0.0001</td>
<td>0.125</td>
<td>1/8</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0.00001</td>
<td>0.0625</td>
<td>1/16</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0.000001</td>
<td>0.03125</td>
<td>1/32</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0.0000001</td>
<td>0.015625</td>
<td>1/64</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0.00000001</td>
<td>0.0078125</td>
<td>1/128</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>0.000000001</td>
<td>0.00390625</td>
<td>1/256</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>0.0000000001</td>
<td>0.001953125</td>
<td>1/512</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>0.00000000001</td>
<td>0.0009765625</td>
<td>1/1024</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>0.000000000001</td>
<td>0.00048828125</td>
<td>1/2048</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>0.0000000000001</td>
<td>0.000244140625</td>
<td>1/4096</td>
<td></td>
</tr>
</tbody>
</table>
Representation of Fractions with Fixed Point

What about addition and multiplication?

Addition is straightforward:

\[
\begin{array}{c}
01.100 \\
+ 00.100 \\
\hline
10.000
\end{array}
\]

Multiplication a bit more complex:

\[
\begin{array}{c}
01.100 \\
00.100 \\
\hline
01100 \\
00000
\end{array}
\]

Where’s the answer, 0.11? (e.g., 0.5 + 0.25; Need to remember where point is!)
Representation of Fractions

- Our examples used a “fixed” binary point. What we really want is to “float” the binary point to make most effective use of limited bits.
- With floating-point representation, each numeral carries an exponent field recording the whereabouts of its binary point.
- Binary point can be outside the stored bits, so very large and small numbers can be represented.

\[ \ldots 000000.001010100000\ldots \]

Store these bits and keep track of the binary point as 2 places to the left of the MSB.

Any other solution would lose accuracy!

Example: put \(0.1640625_{\text{ten}}\) into binary. Represent with 5-bits choosing where to put the binary point.
Scientific Notation (in Decimal)

- 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: $0.1 \times 10^{-8}$, $10.0 \times 10^{-10}$
Scientific Notation (in Binary)

- 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` and `double` for double precision.
UCB’s “Father” of IEEE Floating point

IEEE Standard 754 for Binary Floating-Point Arithmetic.

1989
ACM Turing Award Winner!

Prof. Kahan

www.cs.berkeley.edu/~wkahan/.../ieee754status/754story.html
Goals for **IEEE 754 Floating-Point Standard**

- Standard arithmetic for reals for all computers
  - Important because computer representation of real numbers is approximate. Want same results on all computers.
- Keep as much precision as possible
- Help programmer with errors in real arithmetic
  - $+\infty$, $-\infty$, Not-A-Number (NaN), exponent overflow, exponent underflow, +/- zero
- 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
Floating-Point Representation

- For “single precision”, a 32-bit word.
- IEEE 754 single precision 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 because leading 1 is implicit)
    $$(-1)^s \times (1 + F) \times 2^E$$
- Can represent approximately numbers in the range of $2.0 \times 10^{-38}$ to $2.0 \times 10^{38}$
Floating-Point Representation

- Normal format: \((-1)^S \times 1.xxx...x \times 2^{(yyy...y - 127)}\)

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>23</th>
<th>22</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>(S)</td>
<td>Exponent</td>
<td>Significand</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- \(S\) represents Sign
  - 1 for negative, 0 for positive
- x’s represent Fractional part called Significand
  - implicit leading 1, signed-magnitude (not 2’s complement)
- y’s represent Exponent
  - in special form called biased notation (stored value - 127)
IEEE 754 Floating-Point Standard

- 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
Bias Notation (exponent = stored value - 127)

<table>
<thead>
<tr>
<th>Decimal Exponent</th>
<th>signed 2’s complement</th>
<th>Biased Notation</th>
<th>Decimal Value of Biased Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>For infinities</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>127</td>
<td>01111111</td>
<td>11111111</td>
<td>255</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>11111110</td>
<td>254</td>
</tr>
<tr>
<td>2</td>
<td>00000010</td>
<td>10000001</td>
<td>129</td>
</tr>
<tr>
<td>1</td>
<td>00000001</td>
<td>10000000</td>
<td>128</td>
</tr>
<tr>
<td>0</td>
<td>00000000</td>
<td>01111111</td>
<td>127</td>
</tr>
<tr>
<td>-1</td>
<td>11111111</td>
<td>01111110</td>
<td>126</td>
</tr>
<tr>
<td>-2</td>
<td>11111110</td>
<td>01111101</td>
<td>125</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>-∞, NaN</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Getting closer to zero

Zero
Floating-Point Representation

• What if result too large?
  (> 2.0\times 10^{38}, < -2.0\times 10^{38} )
  • **Overflow!** ⇒ Exponent larger than represented in 8-bit Exponent field

• What if result too small?
  (>0 & < 2.0\times 10^{-38}, <0 & > -2.0\times 10^{-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

• What about bigger or smaller numbers?
• IEEE 754 Floating-Point **Double Precision Standard** (64 bits)
  • 1 bit for *sign (s)* of floating-point number
  • 11 bits for *exponent (E)*
  • 52 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 $2.0 \times 10^{-308}$ to $2.0 \times 10^{308}$
• More importantly, 53 bits of precision!
• Recall, 32-bit format called **Single Precision**
Peer Instruction

What’s the value of this Floating Point number:
1 1000 0000 1000 0000 0000 0000 0000 0000 000

- A: \(-1 \times 2^{128}\)
- B: \(+1 \times 2^{-128}\)
- C: \(-1 \times 2^{1}\)
- D: \(-1.5 \times 2^{1}\)
Another Clicker Question
Which is Less? (i.e., closer to \(-\infty\))

- A or B
- 0 vs. \(1 \times 10^{-127}\)?
- \(1 \times 10^{-126}\) vs. \(1 \times 10^{-127}\)?
- \(-1 \times 10^{-127}\) vs. 0?
- \(-1 \times 10^{-126}\) vs. \(-1 \times 10^{-127}\)?
Administrivia…

• Project 3/2 due 3/23
  • But spring break is only a single day for lateness
  • Autograder has been released!

• Enjoy spring break!
More Floating Point: Preview

- What about 0?
  - Bit pattern all 0s means 0 (so no implicit leading 1 in this case)
- What if divide 1 by 0?
  - Can get infinity symbols $+\infty$, $-\infty$
  - Sign bit 0 or 1, largest exponent (all 1s), 0 in fraction
- What if do something stupid? ($\infty - \infty$, $0 \div 0$)
  - Can get special symbols NaN for “Not-a-Number”
  - Sign bit 0 or 1, largest exponent (all 1s), not zero in fraction
- What if result is too big?
  - Get overflow in exponent, alert programmer!
- What if result is too small?
  - Get underflow in exponent, alert programmer!
Representation for 0

- Represent 0?
  - Exponent all zeroes
  - Significand all zeroes
  - What about sign? Both cases valid!

\[ +0: \quad 0 \quad 00000000 \quad 00000000000000000000000000000000 \]
\[ -0: \quad 1 \quad 00000000 \quad 00000000000000000000000000000000 \]
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*
### 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>0</td>
<td>nonzero</td>
<td>???</td>
</tr>
<tr>
<td>1-254</td>
<td>anything</td>
<td>Normal Floating Point</td>
</tr>
<tr>
<td>255</td>
<td>0</td>
<td>Infinity</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 $\infty$ 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: $\text{op}(\text{NaN}, X) = \text{NaN}$
  • 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:
    \[ a = 1.0\ldots_2 \times 2^{-126} = 2^{-126} \]
  - Second smallest representable positive number:
    \[ b = 1.000\ldots1_2 \times 2^{-126} = (1 + 0.00\ldots1_2) \times 2^{-126} = (1 + 2^{-23}) \times 2^{-126} = 2^{-126} + 2^{-149} \]
    \[ a - 0 = 2^{-126} \]
    \[ b - a = 2^{-149} \]
    
    Normalization and implicit 1 are to blame!
• Solution:
  – We still haven’t used Exponent = 0, Significand nonzero
  – DEnormalized number: no (implied) leading 1, \(\text{implicit exponent} = -126\)
  – Smallest representable positive number:
    \(a = 2^{-149}\) (i.e., \(2^{-126} \times 2^{-23}\))
  – Second-smallest representable positive number:
    \(b = 2^{-148}\) (i.e., \(2^{-126} \times 2^{-22}\))
# Special Numbers Summary

<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>Normal Floating Point</td>
</tr>
<tr>
<td>255</td>
<td>0</td>
<td>Infinity</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

• Floating Point Standard
• Defining Performance
• And in Conclusion …
What’s important to measure about computer systems?

• As a consumer and user of computer systems, what’s important to you?
  • cost, performance, power consumption
  • reliability, software infrastructure (OS, etc.)

• As a designer of computer systems, what’s important to you?
  • cost, performance, power consumption
  • reliability, software infrastructure, time to market

• Achieving all of low-cost, high-performance, and low-power consumption is not possible:
  • Ex: Very high-performance computers are high-cost and high-power (supercomputers), Low-cost computers are low-performance (Arduino, RP), Low-power computers are low-performance (few watts versus hundreds of watts).

These characteristics “tradeoff” with one another. Improving on one usually comes at the expense of the others.
Review: 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

• 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].
Review of CPU Latency Performance Equation

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

- Total number of instructions *executed* to run program of interest.
- Average number of cycles per instruction
- System clock period \((1/F)\)
What Affects Each Component? Instruction Count, CPI, Clock Rate

<table>
<thead>
<tr>
<th>Hardware or software?</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, CPI, Clock Rate</td>
</tr>
<tr>
<td>Micro-architecture</td>
<td>CPI, Clock Rate</td>
</tr>
</tbody>
</table>

Why is the CPI not determined only by the instruction set architecture and the micro-architecture?
Peer Instruction

• Which computer has the highest performance for the given program?

<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>1.0 GHz</td>
<td>2</td>
<td>1000</td>
</tr>
<tr>
<td>B</td>
<td>2.0 GHz</td>
<td>5</td>
<td>800</td>
</tr>
<tr>
<td>C</td>
<td>0.5 GHz</td>
<td>1.25</td>
<td>400</td>
</tr>
<tr>
<td>D</td>
<td>5.0 GHz</td>
<td>10</td>
<td>2000</td>
</tr>
</tbody>
</table>
Administrivia
Measuring Processor Performance with Standard Benchmarks

• The best way to measure computer system performance is to measure using the set of programs of interest to you, or a set that is representative of your typical "workload".

• Performing accurate measurements is tricky and your future needs might be unknown. Most users rely on standard benchmarks for comparisons.

• Computer designers (and manufacturers) rely on standard benchmarks for understanding the effect of design decisions on performance.

• Popular Benchmark Suites:
  • SPEC - Standard Performance Evaluation Corporation
    • https://www.spec.org/benchmarks.html
  • EEMBC - Industry-Standard Benchmarks for Embedded Systems
    • https://www.eembc.org
  • Many more used in industry and academia …
SPEC (Standard Performance Evaluation Corporation)

- A non-profit organization founded in 1988 to establish standardized performance benchmarks that are objective, meaningful, clearly defined, and readily available.
  - SPEC members include hardware and software vendors, universities, and researchers.
  - SPEC was founded on the realization that "An ounce of honest data is worth a pound of marketing hype".

- SPEC CPU benchmarks focus on compute intensive performance:
  - Processor - The CPU chip(s).
  - Memory - The memory hierarchy, including caches and main memory.
  - Compilers - C, C++, and Fortran compilers, including optimizers.

- SPEC CPU2006
  - 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
## 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 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, When comparing any two measured systems, their relative performance to each other remains the same no matter what computer is used as reference
  - Geometric Mean for Barcelona is 11.7
### SPEC 2017 - Execution-time and Throughput

<table>
<thead>
<tr>
<th>Short Tag</th>
<th>Suite</th>
<th>Contents</th>
<th>Metrics</th>
<th>How many copies? What do Higher Scores Mean?</th>
</tr>
</thead>
<tbody>
<tr>
<td>intspeed</td>
<td>SPECspeed 2017 Integer</td>
<td>10 integer benchmarks</td>
<td>SPECspeed2017_int_base</td>
<td>SPECspeed suites always run one copy of each benchmark. Higher scores indicate that less time is needed.</td>
</tr>
<tr>
<td>fpspeed</td>
<td>SPECspeed 2017 Floating Point</td>
<td>10 floating point benchmarks</td>
<td>SPECspeed2017_fp_base</td>
<td></td>
</tr>
<tr>
<td>intrate</td>
<td>SPECrate 2017 Integer</td>
<td>10 integer benchmarks</td>
<td>SPECrate2017_int_base</td>
<td>SPECrate suites run multiple concurrent copies of each benchmark. The tester selects how many. Higher scores indicate more throughput (work per unit of time).</td>
</tr>
<tr>
<td>fprate</td>
<td>SPECrate 2017 Floating Point</td>
<td>13 floating point benchmarks</td>
<td>SPECrate2017_fp_base</td>
<td></td>
</tr>
</tbody>
</table>
## SPEC CPU2017 Integer Suite

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>500.perlbench_r</td>
<td>600.perlbench_s</td>
<td>C</td>
<td>362</td>
<td>Perl interpreter</td>
</tr>
<tr>
<td>502.gcc_r</td>
<td>602.gcc_s</td>
<td>C</td>
<td>1,304</td>
<td>GNU C compiler</td>
</tr>
<tr>
<td>505.mcf_r</td>
<td>605.mcf_s</td>
<td>C</td>
<td>3</td>
<td>Route planning</td>
</tr>
<tr>
<td>520.omnetpp_r</td>
<td>620.omnetpp_s</td>
<td>C++</td>
<td>134</td>
<td>Discrete Event simulation - computer network</td>
</tr>
<tr>
<td>523.xalanchmk_r</td>
<td>623.xalanchmk_s</td>
<td>C++</td>
<td>520</td>
<td>XML to HTML conversion via XSLT</td>
</tr>
<tr>
<td>525.x264_r</td>
<td>625.x264_s</td>
<td>C</td>
<td>96</td>
<td>Video compression</td>
</tr>
<tr>
<td>531.deepsjeng_r</td>
<td>631.deepsjeng_s</td>
<td>C++</td>
<td>10</td>
<td>Artificial Intelligence: alpha-beta tree search (Chess)</td>
</tr>
<tr>
<td>541.leela_r</td>
<td>641.leela_s</td>
<td>C++</td>
<td>21</td>
<td>Artificial Intelligence: Monte Carlo tree search (Go)</td>
</tr>
<tr>
<td>548.exchange2_r</td>
<td>648.exchange2_s</td>
<td>Fortran</td>
<td>1</td>
<td>Artificial Intelligence: recursive solution generator (Sudoku)</td>
</tr>
<tr>
<td>557.xz_r</td>
<td>657.xz_s</td>
<td>C</td>
<td>33</td>
<td>General data compression</td>
</tr>
</tbody>
</table>
## SPEC CPU2017 Floating Point Suite

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>503.bwaves_r</td>
<td>603.bwaves_s</td>
<td>Fortran</td>
<td>1</td>
<td>Explosion modeling</td>
</tr>
<tr>
<td>507.cactuBSSN_r</td>
<td>607.cactuBSSN_s</td>
<td>C++, C, Fortran</td>
<td>257</td>
<td>Physics: relativity</td>
</tr>
<tr>
<td>508.namd_r</td>
<td></td>
<td>C++</td>
<td>8</td>
<td>Molecular dynamics</td>
</tr>
<tr>
<td>510.parest_r</td>
<td></td>
<td>C++</td>
<td>427</td>
<td>Biomedical imaging: optical tomography with finite elements</td>
</tr>
<tr>
<td>511.povray_r</td>
<td></td>
<td>C++, C</td>
<td>170</td>
<td>Ray tracing</td>
</tr>
<tr>
<td>519.lb m_r</td>
<td>619.lb m_s</td>
<td>C</td>
<td>1</td>
<td>Fluid dynamics</td>
</tr>
<tr>
<td>521.wrf_r</td>
<td>621.wrf_s</td>
<td>Fortran, C</td>
<td>991</td>
<td>Weather forecasting</td>
</tr>
<tr>
<td>526.blender_r</td>
<td></td>
<td>C++, C</td>
<td>1,577</td>
<td>3D rendering and animation</td>
</tr>
<tr>
<td>527.cam4_r</td>
<td>627.cam4_s</td>
<td>Fortran, C</td>
<td>407</td>
<td>Atmosphere modeling</td>
</tr>
<tr>
<td>538.imagick_r</td>
<td>638.imagick_s</td>
<td>Fortran, C</td>
<td>338</td>
<td>Wide-scale ocean modeling (climate level)</td>
</tr>
<tr>
<td>544.nab_r</td>
<td>644.nab_s</td>
<td>C</td>
<td>259</td>
<td>Image manipulation</td>
</tr>
<tr>
<td>549.fotonik3d_r</td>
<td>649.fotonik3d_s</td>
<td>C</td>
<td>24</td>
<td>Molecular dynamics</td>
</tr>
<tr>
<td>554.roms_r</td>
<td>654.roms_s</td>
<td>Fortran</td>
<td>14</td>
<td>Computational Electromagnetics</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>210</td>
<td>Regional ocean modeling</td>
</tr>
</tbody>
</table>
Outline

• Floating Point Standard
• Defining Performance
• And in Conclusion …
• Time (seconds/program) is measure of performance
  \[ \text{Instructions} = \frac{\text{Program}}{\text{Instruction}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock Cycle}} \]

• 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:

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>23</th>
<th>22</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>Exponent</td>
<td>Significand</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 bit</td>
<td>8 bits</td>
<td>23 bits</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>