CS161 – Design and Architecture of Computer Systems

Performance Evaluation
WHAT IS PERFORMANCE?
Understanding Performance

- Algorithm
  - Determines number of operations executed
- Programming language, compiler, architecture
  - Determine number of machine instructions executed per operation
- Processor and memory system
  - Determine how fast instructions are executed
- I/O system (including OS)
  - Determines how fast I/O operations are executed
Response Time and Throughput

- **Response time**
  - How long it takes to do a task

- **Throughput**
  - Total work done per unit time
    - e.g., tasks/transactions/… per hour

- How are response time and throughput affected by
  - Replacing the processor with a faster version?
  - Adding more processors?

- We’ll focus on response time for now…
Relative Performance

- Define Performance = 1/Execution Time
- “X is \( n \) time faster than Y”

\[
\frac{\text{Performance}_X}{\text{Performance}_Y} = \frac{\text{Execution time}_Y}{\text{Execution time}_X} = n
\]

Example: time taken to run a program
- 10s on A, 15s on B
- Execution Time\(_B\) / Execution Time\(_A\)
  - = 15s / 10s = 1.5
- So A is 1.5 times faster than B
Relative Performance

- Define Performance = 1/Execution Time
- “X is \( n \) time faster than Y”

\[
\frac{\text{Performance}_X}{\text{Performance}_Y} = \frac{\text{Execution time}_Y}{\text{Execution time}_X} = n
\]

- Example: time taken to run a program
  - 60s on A, 30s on B
  - Execution Time\(_B\) / Execution Time\(_A\) = 30s / 60s = 0.5
    - So A is 0.5 times faster than B
  - or B is 2 times faster than A
Measuring Execution Time

- Elapsed time
  - Total response time, including all aspects
    - Processing, I/O, OS overhead, idle time
  - Determines system performance

- CPU time
  - Time spent processing a given job
    - Discounts I/O time, other jobs’ shares
  - Comprises user CPU time and system CPU time
  - Different programs are affected differently by CPU and system performance
CPU Clocking

- Operation of digital hardware governed by a constant-rate clock

- Clock period: duration of a clock cycle
  - e.g., 250ps = 0.25ns = 250×10^{-12}s

- Clock frequency (rate): cycles per second
  - e.g., 4.0GHz = 4000MHz = 4.0×10^9Hz
CPU Time

CPU Time = CPU Clock Cycles \times \text{Clock Cycle Time}

\[ \frac{\text{CPU Clock Cycles}}{\text{Clock Rate}} \]

Performance improved by

- Reducing number of clock cycles
- Increasing clock rate
- Hardware designer must often trade off clock rate against cycle count
**CPU Time Example**

- Computer A: 2GHz clock, 10s CPU time
- Designing Computer B
  - Aim for 6s CPU time
  - Can do faster clock, but causes $1.2 \times \text{clock cycles}$
- How fast must Computer B clock be?

<table>
<thead>
<tr>
<th>Rate</th>
<th>Clock</th>
<th>CPU Time</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>2GHz</td>
<td>10s</td>
<td>20G</td>
</tr>
<tr>
<td>B</td>
<td>4GHz</td>
<td>6s</td>
<td>24G</td>
</tr>
</tbody>
</table>

\[
\text{Clock Rate}_B = \frac{\text{Clock Cycles}_B}{\text{CPU Time}_B} = \frac{1.2 \times \text{Clock Cycles}_A}{6s}
\]

\[
\text{Clock Cycles}_A = \text{CPU Time}_A \times \text{Clock Rate}_A
\]

\[
= 10s \times 2GHz = 20 \times 10^9
\]

\[
\text{Clock Rate}_B = \frac{1.2 \times 20 \times 10^9}{6s} = \frac{24 \times 10^9}{6s} = 4GHz
\]
Instruction Count and CPI

Clock Cycles = Instruction Count × Cycles per Instruction

CPU Time = Instruction Count × CPI × Clock Cycle Time

\[
\text{CPU Time} = \frac{\text{Instruction Count} \times \text{CPI}}{\text{Clock Rate}}
\]

- Instruction Count for a program
  - Determined by program, ISA and compiler
- Average cycles per instruction
  - Determined by CPU hardware
  - If different instructions have different CPI
    - Average CPI affected by instruction mix
CPI Example

- Computer A: Cycle Time = 250ps, CPI = 2.0
- Computer B: Cycle Time = 500ps, CPI = 1.2
- Same ISA
- Which is faster, and by how much?

\[
\begin{align*}
\text{CPU Time}_A &= \text{Instruction Count} \times \text{CPI}_A \times \text{Cycle Time}_A \\
&= l \times 2.0 \times 250\text{ps} = l \times 500\text{ps} \\
\text{CPU Time}_B &= \text{Instruction Count} \times \text{CPI}_B \times \text{Cycle Time}_B \\
&= l \times 1.2 \times 500\text{ps} = l \times 600\text{ps}
\end{align*}
\]

\[
\frac{\text{CPU Time}_B}{\text{CPU Time}_A} = \frac{l \times 600\text{ps}}{l \times 500\text{ps}} = 1.2
\]

A is faster…

…by this much
CPI in More Detail

- If different instruction types take different numbers of cycles

\[
\text{Clock Cycles} = \sum_{i=1}^{n} (\text{CPI}_i \times \text{Instruction Count}_i)
\]

- Weighted average CPI

\[
\text{CPI} = \frac{\text{Clock Cycles}}{\text{Instruction Count}} = \sum_{i=1}^{n} \left( \text{CPI}_i \times \frac{\text{Instruction Count}_i}{\text{Instruction Count}} \right)
\]

Relative frequency
CPI Example

Alternative compiled code sequences using instructions in type INT, FP, MEM

<table>
<thead>
<tr>
<th>Type</th>
<th>INT</th>
<th>FP</th>
<th>MEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPI for type</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>IC in Program 1</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>IC in Program 2</td>
<td>4</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

- **Program 1: IC = 5**
  - Clock Cycles
    - $= 2 \times 1 + 1 \times 2 + 2 \times 3$
    - $= 10$
  - Avg. CPI = $10/5 = 2.0$

- **Program 2: IC = 6**
  - Clock Cycles
    - $= 4 \times 1 + 1 \times 2 + 1 \times 3$
    - $= 9$
  - Avg. CPI = $9/6 = 1.5$
Performance Summary

The BIG Picture

CPU Time = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock cycle}}

- Performance depends on
  - Algorithm: affects IC, possibly CPI
  - Programming language: affects IC, CPI
  - Compiler: affects IC, CPI
  - Instruction set architecture: affects IC, CPI, $T_c$
In CMOS IC technology

Power = Capacitive load $\times$ Voltage$^2$ $\times$ Frequency

$\times 30$

$5V \rightarrow 1V$

$\times 1000$
Reducing Power

- Suppose a new CPU has
  - 85% of capacitive load of old CPU
  - 15% voltage and 15% frequency reduction

\[
\frac{P_{\text{new}}}{P_{\text{old}}} = \frac{C_{\text{old}} \times 0.85 \times (V_{\text{old}} \times 0.85)^2 \times F_{\text{old}} \times 0.85}{C_{\text{old}} \times V_{\text{old}}^2 \times F_{\text{old}}} = 0.85^4 = 0.52
\]

- The power wall
  - We can’t reduce voltage further
  - We can’t remove more heat

- How else can we improve performance?
Multiprocessors

- Multicore microprocessors
  - More than one processor per chip
- Requires explicitly parallel programming
  - Compare with instruction level parallelism
    - Hardware executes multiple instructions at once
    - Hidden from the programmer
- Hard to do
  - Programming for performance
  - Load balancing
  - Optimizing communication and synchronization
AMD Opteron X2 Wafer

- X2: 300mm wafer, 117 chips, 90nm technology
- X4: 45nm technology
Manufacturing ICs

Yield: proportion of working dies per wafer
Integrated Circuit Cost

Cost per die = \frac{\text{Cost per wafer}}{\text{Dies per wafer} \times \text{Yield}}

Dies per wafer \approx \frac{\text{Wafer area}}{\text{Die area}}

Yield = \frac{1}{\left(1 + (\text{Defects per area} \times \frac{\text{Die area}}{2})\right)^2}

- Nonlinear relation to area and defect rate
- Wafer cost and area are fixed
- Defect rate determined by manufacturing process
- Die area determined by architecture and circuit design
SPEC CPU Benchmark

- Programs used to measure performance
  - Supposedly typical of actual workload
- Standard Performance Evaluation Corp (SPEC)
  - Develops benchmarks for CPU, I/O, Web, ...
- SPEC CPU2006
  - Elapsed time to execute a selection of programs
    - Negligible I/O, so focuses on CPU performance
  - Normalize relative to reference machine
  - Summarize as geometric mean of performance ratios
    - CINT2006 (integer) and CFP2006 (floating-point)

\[
\sqrt[n]{\prod_{i=1}^{n} \text{Execution time ratio}_i}
\]
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>IC×10⁹</th>
<th>CPI</th>
<th>Tc (ns)</th>
<th>Exec time</th>
<th>Ref time</th>
<th>SPECratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>perl</td>
<td>Interpreted string processing</td>
<td>2,118</td>
<td>0.75</td>
<td>0.40</td>
<td>637</td>
<td>9,777</td>
<td>15.3</td>
</tr>
<tr>
<td>bzip2</td>
<td>Block-sorting compression</td>
<td>2,389</td>
<td>0.85</td>
<td>0.40</td>
<td>817</td>
<td>9,650</td>
<td>11.8</td>
</tr>
<tr>
<td>gcc</td>
<td>GNU C Compiler</td>
<td>1,050</td>
<td>1.72</td>
<td>0.47</td>
<td>24</td>
<td>8,050</td>
<td>11.1</td>
</tr>
<tr>
<td>mcf</td>
<td>Combinatorial optimization</td>
<td>336</td>
<td>10.00</td>
<td>0.40</td>
<td>1,345</td>
<td>9,120</td>
<td>6.8</td>
</tr>
<tr>
<td>go</td>
<td>Go game (AI)</td>
<td>1,658</td>
<td>1.09</td>
<td>0.40</td>
<td>721</td>
<td>10,490</td>
<td>14.6</td>
</tr>
<tr>
<td>hmer</td>
<td>Search gene sequence</td>
<td>2,783</td>
<td>0.80</td>
<td>0.40</td>
<td>890</td>
<td>9,330</td>
<td>10.5</td>
</tr>
<tr>
<td>sjeng</td>
<td>Chess game (AI)</td>
<td>2,176</td>
<td>0.96</td>
<td>0.48</td>
<td>37</td>
<td>12,100</td>
<td>14.5</td>
</tr>
<tr>
<td>libquantum</td>
<td>Quantum computer simulation</td>
<td>1,623</td>
<td>1.61</td>
<td>0.40</td>
<td>1,047</td>
<td>20,720</td>
<td>19.8</td>
</tr>
<tr>
<td>h264avc</td>
<td>Video compression</td>
<td>3,102</td>
<td>0.80</td>
<td>0.40</td>
<td>993</td>
<td>22,130</td>
<td>22.3</td>
</tr>
<tr>
<td>omnetpp</td>
<td>Discrete event simulation</td>
<td>587</td>
<td>2.94</td>
<td>0.40</td>
<td>690</td>
<td>6,250</td>
<td>9.1</td>
</tr>
<tr>
<td>astar</td>
<td>Games/path finding</td>
<td>1,082</td>
<td>1.79</td>
<td>0.40</td>
<td>773</td>
<td>7,020</td>
<td>9.1</td>
</tr>
<tr>
<td>xalancbmk</td>
<td>XML parsing</td>
<td>1,058</td>
<td>2.70</td>
<td>0.40</td>
<td>1,143</td>
<td>6,900</td>
<td>6.0</td>
</tr>
<tr>
<td>Geometric mean</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>11.7</td>
</tr>
</tbody>
</table>

High cache miss rates
SPEC Power Benchmark

- Power consumption of server at different workload levels
  - Performance: ssj_ops/sec
  - Power: Watts (Joules/sec)

Overall ssj_ops per Watt = \[ \frac{\sum_{i=0}^{10} ssj\_ops_i}{\sum_{i=0}^{10} power_i} \]
### SPECpower_ssj2008 for X4

<table>
<thead>
<tr>
<th>Target Load %</th>
<th>Performance (ssj_ops/sec)</th>
<th>Average 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>920,35</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>Overall sum</td>
<td>1,283,590</td>
<td>2,605</td>
</tr>
<tr>
<td>( \sum \text{ssj_ops}/ \sum \text{power} )</td>
<td></td>
<td>493</td>
</tr>
</tbody>
</table>
Fallacy: Low Power at Idle

- Look back at X4 power benchmark
  - At 100% load: 295W
  - At 50% load: 246W (83%)
  - At 10% load: 180W (61%)

- Google data center
  - Mostly operates at 10% – 50% load
  - At 100% load less than 1% of the time

- Consider designing processors to make power proportional to load
Pitfall: Amdahl’s Law

- Improving an aspect of a computer and expecting a proportional improvement in overall performance

\[
T_{\text{improved}} = \frac{T_{\text{affected}}}{\text{improvement factor}} + T_{\text{unaffected}}
\]

- Example: multiply accounts for 80s/100s
  - How much improvement in multiply performance to get 5× overall?

\[
20 = \frac{80}{n} + 20
\]

- Can’t be done!

- Corollary: make the common case fast
Pitfall: MIPS as a Performance Metric

- MIPS: Millions of Instructions Per Second
  - Doesn’t account for
    - Differences in ISAs between computers
    - Differences in complexity between instructions

\[
\text{MIPS} = \frac{\text{Instruction count}}{\text{Execution time} \times 10^6} = \frac{\text{Instruction count}}{\text{Instruction count} \times \text{CPI}} \times \frac{\text{Clock rate}}{\text{CPI} \times 10^6}
\]

- CPI varies between programs on a given CPU
Concluding Remarks

- Cost/performance is improving
  - Due to underlying technology development
- Hierarchical layers of abstraction
  - In both hardware and software
- Instruction set architecture
  - The hardware/software interface
- Execution time: the best performance measure
- Power is a limiting factor
  - Use parallelism to improve performance