Chapter 1

Computer Abstractions and Technology

The Computer Revolution

- Progress in computer technology
  - Underpinned by Moore's Law
- Makes novel applications feasible
  - Computers in automobiles
  - Cell phones
  - Human genome project
  - World Wide Web
  - Search Engines
- Computers are pervasive

Classes of Computers

- Personal computers
  - General purpose, variety of software
  - Subject to cost/performance tradeoff
- Server computers
  - Network based
  - High capacity, performance, reliability
  - Range from small servers to building sized
Classes of Computers

- **Supercomputers**
  - High-end scientific and engineering calculations
  - Highest capability but represent a small fraction of the overall computer market

- **Embedded computers**
  - Hidden as components of systems
  - Stringent power/performance/cost constraints

The PostPC Era

- **Personal Mobile Device (PMD)**
  - Battery operated
  - Connects to the Internet
  - Hundreds of dollars
  - Smart phones, tablets, electronic glasses

- **Cloud computing**
  - Warehouse Scale Computers (WSC)
  - Software as a Service (SaaS)
  - Portion of software run on a PMD and a portion run in the Cloud
  - Amazon and Google
What You Will Learn

- How programs are translated into the machine language
- And how the hardware executes them
- The hardware/software interface
- What determines program performance
- And how it can be improved
- How hardware designers improve performance
- What is parallel processing

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

Eight Great Ideas

- Design for Moore's Law
- Use abstraction to simplify design
- Make the common case fast
- Performance via parallelism
- Performance via pipelining
- Performance via prediction
- Hierarchy of memories
- Dependability via redundancy
Below Your Program

- Application software
  - Written in high-level language
- System software
  - Compiler: translates HLL code to machine code
  - Operating System: service code
    - Handling input/output
    - Managing memory and storage
    - Scheduling tasks & sharing resources
- Hardware
  - Processor, memory, I/O controllers

Levels of Program Code

- High-level language
  - Level of abstraction closer to problem domain
  - Provides for productivity and portability
- Assembly language
  - Textual representation of instructions
- Hardware representation
  - Binary digits (bits)
  - Encoded instructions and data

Components of a Computer

The BIG Picture

Same components for all kinds of computer
- Desktop, server, embedded

Input/output includes
- User-interface devices
  - Display, keyboard, mouse
- Storage devices
  - Hard disk, CD/DVD, flash
- Network adapters
  - For communicating with other computers
**Touchscreen**

- PostPC device
- Supersedes keyboard and mouse
- Resistive and Capacitive types
  - Most tablets, smart phones use capacitive
  - Capacitive allows multiple touches simultaneously

**Through the Looking Glass**

- LCD screen: picture elements (pixels)
  - Mirrors content of frame buffer memory

**Opening the Box**

- Capacitive multitouch LCD screen
- 3.8 V, 25 Watt-hour battery
- Computer board
**Inside the Processor (CPU)**
- Datapath: performs operations on data
- Control: sequences datapath, memory, ...
- Cache memory
  - Small fast SRAM memory for immediate access to data

**Inside the Processor**
- Apple A5

**Abstractions**

**The BIG Picture**
- Abstraction helps us deal with complexity
  - Hide lower-level detail
- Instruction set architecture (ISA)
  - The hardware/software interface
- Application binary interface
  - The ISA plus system software interface
- Implementation
  - The details underlying and interface
A Safe Place for Data

- Volatile main memory
  - Loses instructions and data when power off
- Non-volatile secondary memory
  - Magnetic disk
  - Flash memory
  - Optical disk (CDROM, DVD)

Networks

- Communication, resource sharing, nonlocal access
- Local area network (LAN): Ethernet
- Wide area network (WAN): the Internet
- Wireless network: WiFi, Bluetooth

Technology Trends

- Electronics technology continues to evolve
  - Increased capacity and performance
  - Reduced cost

<table>
<thead>
<tr>
<th>Year</th>
<th>Technology</th>
<th>Relative performance/cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>1951</td>
<td>Vacuum tube</td>
<td>1</td>
</tr>
<tr>
<td>1965</td>
<td>Transistor</td>
<td>35</td>
</tr>
<tr>
<td>1975</td>
<td>Integrated circuit (IC)</td>
<td>500</td>
</tr>
<tr>
<td>1985</td>
<td>Very large scale IC (VLSI)</td>
<td>2,400,000</td>
</tr>
<tr>
<td>2013</td>
<td>Ultra large scale IC</td>
<td>250,000,000,000</td>
</tr>
</tbody>
</table>
**Semiconductor Technology**

- Silicon: semiconductor
- Add materials to transform properties:
  - Conductors
  - Insulators
  - Switch

**Manufacturing ICs**

![IC Manufacturing Process Diagram]

- Yield: proportion of working dies per wafer

**Intel Core i7 Wafer**

- 300mm wafer, 280 chips, 32nm technology
- Each chip is 20.7 x 10.5 mm
Integrated Circuit Cost

- Cost per die = \( \frac{\text{Cost per wafer}}{\text{Dies per wafer} \times \text{Yield}} \)
- \( \text{Dies per wafer} = \frac{\text{Wafer area}}{\text{Die area}} \)
- \( \text{Yield} = \left(1 + \left(\frac{\text{Defects per area} \times \text{Die area}}{2}\right)\right)^{-1} \)
- 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

Defining Performance

- Which airplane has the best performance?

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”
    \[
    \text{Performance}_X / \text{Performance}_Y = \text{Execution time}_Y / \text{Execution time}_X = n
    \]
  - 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

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 × Clock Cycle Time

- 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 × clock cycles
- How fast must Computer B clock be?

  \[
  \text{Clock Rate}_B = \frac{\text{Clock Cycles}_A}{\text{CPU Time}_A} = \frac{1.2 \times \text{Clock Cycles}_A}{6s} = \frac{1.2 \times 20 \times 10^9}{6s} = 24 \times 10^9 
  \]

  \[
  \text{Clock Rate}_B = \frac{24 \times 10^9}{6s} = 4GHz 
  \]

Instruction Count and CPI

Clock Cycles = Instruction Count × Cycles per Instruction

CPU Time = Instruction Count × CPI × Clock Cycle Time

- 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?

\[
\text{CPU Time}_A = \text{Instruction Count}_A \times \text{CPI}_A \times \text{Cycle Time}_A
\]
\[= 1 \times 2.0 \times 250\text{ps} = 1 \times 500\text{ps} \quad \rightarrow \quad \text{A is faster...}
\]

\[
\text{CPU Time}_B = \text{Instruction Count}_B \times \text{CPI}_B \times \text{Cycle Time}_B
\]
\[= 1 \times 1.2 \times 500\text{ps} = 1 \times 600\text{ps}
\]

\[
\frac{\text{CPU Time}_B}{\text{CPU Time}_A} = \frac{1 \times 600\text{ps}}{1 \times 500\text{ps}} = 1.2 \quad \rightarrow \quad \text{by this much}
\]

CPI in More Detail
- If different instruction classes take different numbers of cycles

\[
\text{Clock Cycles} = \sum_i (\text{CPI}_i \times \text{Instruction Count}_i)
\]

- Weighted average CPI

\[
\text{CPI} = \frac{\text{Clock Cycles}}{\text{Instruction Count}} = \frac{\sum_i (\text{CPI}_i \times \text{Instruction Count}_i)}{\sum_i \text{Instruction Count}_i}
\]

CPI Example
- Alternative compiled code sequences using instructions in classes A, B, C

<table>
<thead>
<tr>
<th>Class</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPI for class</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>IC in sequence 1</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>IC in sequence 2</td>
<td>4</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

- Sequence 1: IC = 5
  - Clock Cycles = \(2 \times 1 + 1 \times 2 + 2 \times 3 = 10\)
  - Avg. CPI = \(10/5 = 2.0\)

- Sequence 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, \( T_c \)

Power Trends

- In CMOS IC technology
  
  \[ \text{Power} = \text{Capacitive load} \times \text{Voltage}^2 \times \text{Frequency} \]

Reducing Power

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

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

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

- How else can we improve performance?
§1.8 The Sea Change: The Switch to Multiprocessors

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

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)

Execution time ratio, $\frac{t_i}{t_o}$
## CINT2006 for Intel Core i7 920

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Score</th>
<th>Instruc. Time (ns)</th>
<th>Entropy</th>
<th>Execution Time (ns)</th>
<th>Reference Time (ns)</th>
<th>Variability</th>
</tr>
</thead>
<tbody>
<tr>
<td>FPU</td>
<td>255</td>
<td>143.7</td>
<td>0.05</td>
<td>159</td>
<td>193</td>
<td>3.4%</td>
</tr>
<tr>
<td>MMX</td>
<td>264</td>
<td>160</td>
<td>0.07</td>
<td>170</td>
<td>210</td>
<td>3.8%</td>
</tr>
<tr>
<td>SSE2</td>
<td>259</td>
<td>155</td>
<td>0.06</td>
<td>165</td>
<td>205</td>
<td>3.7%</td>
</tr>
</tbody>
</table>

## SPEC Power Benchmark

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

Overall ssj_ops per Watt = \( \frac{\sum ssj_{ops}}{\sum power} \)

## SPECpower_ssj2008 for Xeon X5650

<table>
<thead>
<tr>
<th>Target Load %</th>
<th>Performance (ssj_ops)</th>
<th>Average Power (Watts)</th>
</tr>
</thead>
<tbody>
<tr>
<td>100%</td>
<td>968,682</td>
<td>298</td>
</tr>
<tr>
<td>80%</td>
<td>795,558</td>
<td>242</td>
</tr>
<tr>
<td>90%</td>
<td>606,051</td>
<td>224</td>
</tr>
<tr>
<td>70%</td>
<td>607,826</td>
<td>214</td>
</tr>
<tr>
<td>60%</td>
<td>521,081</td>
<td>185</td>
</tr>
<tr>
<td>50%</td>
<td>526,757</td>
<td>170</td>
</tr>
<tr>
<td>40%</td>
<td>345,959</td>
<td>157</td>
</tr>
<tr>
<td>30%</td>
<td>202,071</td>
<td>140</td>
</tr>
<tr>
<td>20%</td>
<td>176,071</td>
<td>135</td>
</tr>
<tr>
<td>10%</td>
<td>86,784</td>
<td>121</td>
</tr>
<tr>
<td>0%</td>
<td>0</td>
<td>80</td>
</tr>
<tr>
<td>Overall Sum</td>
<td>4,767,156</td>
<td>1,422</td>
</tr>
</tbody>
</table>

\[ \text{CINT_ssj2008} = 2,950 \]
Pitfall: Amdahl’s Law
- Improving an aspect of a computer and expecting a proportional improvement in overall performance
  \[ T_{\text{improved}} = \frac{T_{\text{improved}}}{T_{\text{improved}} + T_{\text{unaffected}}} \]
- Example: multiply accounts for 80s/100s
  - How much improvement in multiply performance to get 5x overall?
    \[ 20 = \frac{80}{n} + 20 \quad \text{Can’t be done!} \]
- Corollary: make the common case fast

Fallacy: Low Power at Idle
- Look back at i7 power benchmark
  - At 100% load: 258W
  - At 50% load: 170W (66%)
  - At 10% load: 121W (47%)
- 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: 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
  \[ \text{MIPS} = \frac{\text{Instruction count} \times 10^8}{\text{Execution time} \times 10^8} \]
  \[ \text{MIPS} = \frac{\text{Instruction count} \times \text{CPI} \times 10^8}{\text{Clock rate} \times \text{CPI} \times 10^8} \]
  - 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