Computer Abstractions and Technology

Euiseong Seo (euiseong@skku.edu)
Computer Systems Laboratory
Sungkyunkwan University
http://csl.skku.edu
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

- **Desktop 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

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

The graph shows the processor market from 1997 to 2007, with categories for Cell Phones, PCs, and TVs. The vertical axis represents the market values, and the horizontal axis represents the years from 1997 to 2007. The market values for Cell Phones, PCs, and TVs are indicated by different shades of blue bars.
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
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
Anatomy of a Computer

Output device

Input device

Network cable

Input device

Input device
Anatomy of a Mouse

- **Optical mouse**
  - LED illuminates desktop
  - Small low-res camera
  - Basic image processor
    - Looks for x, y movement
  - Buttons & wheel

- **Supersedes roller-ball mechanical mouse**
Through the Looking Glass

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

[Diagram showing various components inside a computer's open case, including:
- Hard drive
- Processor
- Fan with cover
- Spot for memory DIMMs
- Spot for battery
- Motherboard
- Fan with cover
- DVD drive]
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**

- **AMD Barcelona**: 4 processor cores
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 and resource sharing
- Local area network (LAN): Ethernet
  - Within a building
- 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>900</td>
</tr>
<tr>
<td>1995</td>
<td>Very large scale IC (VLSI)</td>
<td>2,400,000</td>
</tr>
<tr>
<td>2005</td>
<td>Ultra large scale IC</td>
<td>6,200,000,000</td>
</tr>
</tbody>
</table>
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”

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

- Example: time taken to run a program
  - 10s on A, 15s on B
  - Execution Time\(_B\) / Execution Time\(_A\)
    \[
    = \frac{15\text{s}}{10\text{s}} = 1.5
    \]
  - 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

= CPU Clock Cycles
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 $1.2 \times$ clock cycles
- How fast must Computer B clock be?

\[
\begin{align*}
\text{Clock Rate}_B &= \frac{\text{Clock Cycles}_B}{\text{CPU Time}_B} = \frac{1.2 \times \text{Clock Cycles}_A}{6s} \\
\text{Clock Cycles}_A &= \text{CPU Time}_A \times \text{Clock Rate}_A \\
&= 10s \times 2GHz = 20 \times 10^9 \\
\text{Clock Rate}_B &= \frac{1.2 \times 20 \times 10^9}{6s} = \frac{24 \times 10^9}{6s} = 4GHz
\end{align*}
\]
Instruction Count and CPI

- **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

Clock Cycles = Instruction Count × Cycles per Instruction

CPU Time = Instruction Count × CPI × Clock Cycle Time

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

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

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

A is faster…

…by this much
CPI in More Detail

- If different instruction classes take different numbers of cycles

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

- Weighted average CPI

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

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

- **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

#### §1.5 The Power Wall

In CMOS IC technology

\[
\text{Power} = \text{Capacitive load} \times \text{Voltage}^2 \times \text{Frequency}
\]

- \times 30
- 5V \rightarrow 1V
- \times 1000
Reduction 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?
Constrained by power, instruction-level parallelism, memory latency
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
Manufacturing ICs

- Yield: proportion of working dies per wafer
AMD Opteron X2 Wafer

- **X2**: 300mm wafer, 117 chips, 90nm technology
- **X4**: 45nm technology
Integrated Circuit Cost

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

Dies per wafer \( \approx \) Wafer area/Die area

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

- 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)
### CINT2006 for Opteron X4 2356

<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>hmmer</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>
</tbody>
</table>

**Geometric mean**  

11.7

---

**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 = \( \frac{\sum_{i=0}^{10} \text{ssj\_ops}_i}{\sum_{i=0}^{10} \text{power}_i} \)
## 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>
</tbody>
</table>

\[
\frac{\sum \text{ssj_ops}}{\sum \text{power}} = 493
\]
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
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: 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

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

- 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