Chapter 1

Computer Abstractions and Technology
What is Computer Architecture?

• The design of computer systems

• Goal: improve “performance”:
  – Run programs faster
    • Execution time: e.g. less waiting time, more simultaneous tasks
    • Throughput: e.g. higher framerate, faster downloads
  – Use less power, last longer on battery power
  – Generate less or more uniformly-distributed heat
  – Handle more secure encryption standards
  – Software defined networking at higher line speeds
  – More scalable
  – Less expensive
Computer Architecture

- Instruction Set Architecture ("architecture")
  - The native programming language of a processor
    - Assembly language
    - Machine language
  - Openly published to users, licensed for chip makers

- Microarchitecture
  - The internal organization of a processor
  - Executes programs
  - Trade secret
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

- **Instruction Set Architecture (ISA):**
  - Assembly code / machine code “language”

- **Microarchitecture:**
  - ISA implementation
Abstraction

- **Abstraction** used to manage complexity of design
  - Hide details that are not important

```
Program code
  ↓
Machine
  ↓
Instructions
  ↓
Datapaths
  ↓
Logic gates
  ↓
Devices (Transistors)
```
Why Study Computer Architecture

- Compiling “machine agnostic” code:
  - Generally achieve ~1-20% of peak theoretical performance
  - Performance tuned code must be explicitly written for the underlying architecture
  - Especially for **embedded** and **special purpose** processors
  - Understanding computer architecture allows for customization:
    - Multicore
    - More efficient use of registers, instructions

- Device drivers must directly interface with peripherals
  - Uses CPU-specific, bit-level features to communicate
  - E.g. memory-mapped I/O, interrupts, DMA, double buffering, bit fields in status/control registers, memory management, virtual memory
Domains and Levels of Modeling

- Structural
- Functional
- Geometric

High level of abstraction

Low level of abstraction
Functional Abstraction

For i=0 to 10
C = C + A[i]

MAR <= PC, memory_read <= 1
PC <= PC + 1
wait until ready = 1
IR <= memory_data
memory_read <= 0

C = XY + (X⊕Y)Z
S = X⊕Y⊕Z

\[ \frac{\partial}{\partial x} V(x,t) = -L \frac{\partial}{\partial t} I(x,t) \]
Structural Abstraction
Semiconductors

- Silicon is a group IV element
- Forms covalent bonds with four neighbor atoms (3D cubic crystal lattice)
- Si is a poor conductor, but conduction characteristics may be altered
- Add impurities/dopants replaces silicon atom in lattice
  - Adds two different types of charge carriers

Spacing = .543 nm
Feature Size

- Shrink minimum feature size...
  - Smaller L decreases carrier time and increases current
  - Therefore, W may also be reduced for fixed current
  - $C_g$, $C_s$, and $C_d$ are reduced
  - Transistor switches faster ($\sim$linear relationship)
# Minimum Feature Size

<table>
<thead>
<tr>
<th>Year</th>
<th>Processor</th>
<th>Performance</th>
<th>Transistor Size</th>
<th>Transistors</th>
</tr>
</thead>
<tbody>
<tr>
<td>1982</td>
<td>i286</td>
<td>6 - 25 MHz</td>
<td>1.5 μm</td>
<td>~134,000</td>
</tr>
<tr>
<td>1986</td>
<td>i386</td>
<td>16 – 40 MHz</td>
<td>1 μm</td>
<td>~270,000</td>
</tr>
<tr>
<td>1989</td>
<td>i486</td>
<td>16 - 133 MHz</td>
<td>.8 μm</td>
<td>~1 million</td>
</tr>
<tr>
<td>1993</td>
<td>Pentium</td>
<td>60 - 300 MHz</td>
<td>.6 μm</td>
<td>~3 million</td>
</tr>
<tr>
<td>1995</td>
<td>Pentium Pro</td>
<td>150 - 200 MHz</td>
<td>.5 μm</td>
<td>~4 million</td>
</tr>
<tr>
<td>1997</td>
<td>Pentium II</td>
<td>233 - 450 MHz</td>
<td>.35 μm</td>
<td>~5 million</td>
</tr>
<tr>
<td>1999</td>
<td>Pentium III</td>
<td>450 – 1400 MHz</td>
<td>.25 μm</td>
<td>~10 million</td>
</tr>
<tr>
<td>2000</td>
<td>Pentium 4</td>
<td>1.3 – 3.8 GHz</td>
<td>.18 μm</td>
<td>~50 million</td>
</tr>
<tr>
<td>2005</td>
<td>Pentium D</td>
<td>2 threads/package</td>
<td>.09 μm</td>
<td>~200 million</td>
</tr>
<tr>
<td>2006</td>
<td>Core 2</td>
<td>2 threads/die</td>
<td>.065 μm</td>
<td>~300 million</td>
</tr>
<tr>
<td>2008</td>
<td>“Nehalem”</td>
<td>8 threads/die</td>
<td>.045 μm</td>
<td>~800 million</td>
</tr>
<tr>
<td>2009</td>
<td>“Westmere”</td>
<td>8 threads/die</td>
<td>.045 μm</td>
<td>~1 billion</td>
</tr>
<tr>
<td>2011</td>
<td>“Sandy Bridge”</td>
<td>12 threads/die</td>
<td>.032 μm</td>
<td>~1.2 billion</td>
</tr>
<tr>
<td>2013</td>
<td>“Ivy Bridge”</td>
<td>16 threads/die</td>
<td>.022 μm</td>
<td>~1.4 billion</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Year</th>
<th>Processor</th>
<th>Speed</th>
<th>Transistor Size</th>
<th>Transistors</th>
</tr>
</thead>
<tbody>
<tr>
<td>2008</td>
<td>NVIDIA Tesla (GT200)</td>
<td>240 threads/die</td>
<td>.065 μm</td>
<td>1.4 billion</td>
</tr>
<tr>
<td>2010</td>
<td>NVIDIA Fermi (GF110)</td>
<td>512 threads/die</td>
<td>.040 μm</td>
<td>3.0 billion</td>
</tr>
<tr>
<td>2012</td>
<td>NVIDIA Kepler (GK104)</td>
<td>1536 threads/die</td>
<td>.028 μm</td>
<td>3.5 billion</td>
</tr>
</tbody>
</table>

---

Chapter 1 — Computer Abstractions and Technology — 13
Inside the Processor

- Apple A5
IC Fabrication

1. Silicon ingot → Slicer → Blank wafers → 20 to 40 processing steps
2. Tested dies → Dicer → Tested wafer → Wafer tester → Patterned wafers
3. Bond die to package → Packaged dies
4. Tested packaged dies → Part tester → Ship to customers
Si Wafer
8” Wafer
8” Wafer

- 8 inch (200 mm) wafer containing Pentium 4 processors
  - 165 dies, die area = 250 mm², 55 million transistors, .18µm
Intel Core i7 Wafer

- 300mm wafer, 280 chips, 32nm technology
- Each chip is 20.7 x 10.5 mm
Speedup / 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
  - \( \frac{\text{Execution Time}_B}{\text{Execution Time}_A} = \frac{15s}{10s} = 1.5 \)
  - So A is 1.5 times faster than B
Integrated Circuit Cost

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

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

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

- 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
Example

- Assume $C$ defects per area and a die area of $D$. Calculate the improvement in yield if the number of defects is reduced by 1.5.

\[
yield_{\text{new}} \div \yield_{\text{orig}} = \frac{1}{\left(1 + \frac{C}{1.5}\right)^2} = \frac{\left(1 + \frac{C}{1.5}\right)^2}{\left(1 + C \cdot \frac{D}{2}\right)^2}
\]

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

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

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

\[
\begin{align*}
\text{Cost per die} &= \frac{\text{Cost per wafer}}{\text{Dies per wafer} \times \text{Yield}} \\
\text{Dies per wafer} &\approx \frac{\text{Wafer area}}{\text{Die area}} \\
\text{Yield} &= \frac{1}{(1 + \text{Defects per area} \times \text{Die area}/2)^2} \\
1 + C D + \frac{C^2 D^2}{4} &= \frac{1 + C D + \frac{C^2 D^2}{9}}{1 + \frac{C D}{1.5} + \frac{C^2 D^2}{9}}
\end{align*}
\]
Example

• Assume a 20 cm diameter wafer has a cost of 15, contains 100 dies, and has 0.031 defects/cm\(^2\).

1. If the number of dies per wafer is increased by 10% and the defects per area unit increases by 15%, find the die area and yield.

\[
\text{die area}_{20\text{cm}} = \frac{\text{wafer area}}{\text{dies per wafer}} = \frac{\pi \times 10^2}{(100 \times 1.1)} = 2.86 \text{ cm}\(^2\)
\]

\[
\text{yield}_{20\text{cm}} = \frac{1}{(1 + (0.03 \times 1.15 \times 2.86/2))^2} = 0.9082
\]

2. Assume a fabrication process improves the yield from 0.92 to 0.95. Find the defects per area unit for each version of the technology given a die area of 200 mm\(^2\).

\[
\text{defects per area}_{0.92} = \frac{(1 - y^{.5})}{(y^{.5} \times \text{die_area}/2)} = \frac{(1-0.92^{.5})}{(0.92^{.5} \times 2/2)} = 0.043 \text{ defects/cm}\(^2\)}
\]

\[
\text{defects per area}_{0.95} = \frac{(1 - y^{.5})}{(y^{.5} \times \text{die_area}/2)} = \frac{(1-0.95^{.5})}{(0.95^{.5} \times 2/2)} = 0.026 \text{ defects/cm}\(^2\)}
\]
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...
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^{9} Hz
CPU Time

CPU Time = CPU Clock Cycles × Clock Cycle Time

= \frac{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?

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

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

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

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

\[
\text{Clock Cycles} = \text{Instruction Count} \times \text{Cycles per Instruction}
\]

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

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

- 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?

\[
\text{CPU Time}_{A} = \text{Instruction Count} \times \text{CPI}_{A} \times \text{Cycle Time}_{A}
\]
\[
= I \times 2.0 \times 250\text{ps} = I \times 500\text{ps}
\]
\[
\text{CPU Time}_{B} = \text{Instruction Count} \times \text{CPI}_{B} \times \text{Cycle Time}_{B}
\]
\[
= I \times 1.2 \times 500\text{ps} = I \times 600\text{ps}
\]
\[
\frac{\text{CPU Time}_{B}}{\text{CPU Time}_{A}} = \frac{I \times 600\text{ps}}{I \times 500\text{ps}} = 1.2
\]

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

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

Relative frequency
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

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, $T_c$
Example

• Suppose one machine, A, executes a program with an average CPI of 2.1
• Suppose another machine, B (with the same instruction set and an enhanced compiler), executes the same program with 25% less instructions and with a CPI of 1.8 at 800MHz

In order for the two machines to have the same performance, what does the clock rate of the first machine (machine A) need to be?

\[
\frac{I_A \cdot CPI_A}{R_A} = \frac{I_B \cdot CPI_B}{R_B}
\]

\[
\frac{I_A \cdot 2.1}{R_A} = \frac{0.75I_A \cdot 1.8}{800 \cdot 10^6}
\]
Example

Suppose a program has the following instruction classes, CPIs, and mixtures:

<table>
<thead>
<tr>
<th>Instruction type</th>
<th>CPI</th>
<th>ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1.4</td>
<td>55%</td>
</tr>
<tr>
<td>B</td>
<td>2.4</td>
<td>15%</td>
</tr>
<tr>
<td>C</td>
<td>2</td>
<td>30%</td>
</tr>
</tbody>
</table>

Your engineers give you the following options:

Option A: Reduce the CPI of instruction type A to 1.1
Option B: Reduce the CPI of instruction type B to 1.2

Which option would you choose and why?

\[
CPI_A = .55(1.1) + .15(2.4) + .30(2) = 1.565
\]

\[
CPI_B = .55(1.4) + .15(1.2) + .30(2) = 1.550
\]
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
Example

- Consider two different implementations, M1 and M2, of the same instruction set. There are three classes of instructions (A, B, and C) in the instruction set. M1 has a clock rate of 800 MHz and M2 has a clock rate of 2 GHz. The average number of cycles for each instruction class and their frequencies (for a typical program) are as follows:

<table>
<thead>
<tr>
<th>Instruction class</th>
<th>Machine M1 CPI</th>
<th>Frequency</th>
<th>Machine M2 CPI</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>50%</td>
<td>2</td>
<td>60%</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>20%</td>
<td>3</td>
<td>30%</td>
</tr>
<tr>
<td>C</td>
<td>4</td>
<td>30%</td>
<td>4</td>
<td>10%</td>
</tr>
</tbody>
</table>

Calculate the average CPI for each machine, M1, and M2.
- M1 => \(0.5(1) + 0.2(2) + 0.3(4) = 2.1\)
- M2 => \(0.6(2) + 0.3(3) + 0.1(4) = 2.5\)

Calculate the average MIPS ratings for each machine, M1 and M2.
Hint: MIPS = (clock rate / CPI) / 10^6.
- M1 => \(800 \times 2.1 = 1680\)
- M2 => \(2000 \times 2.5 = 5000\)
Example (Con’t)

- How many less instructions would M1 need to execute to match the speed of M2?

  - M1 => 800 * 2.1 = 1680
  - M2 => 2000 * 2.5 = 5000

  \[
  \frac{5000}{1680}
  \]
Power Trends

- In CMOS IC technology

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

\[
\times 30 \quad 5V \rightarrow 1V \quad \times 1000
\]
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?
Example

- Assume:

<table>
<thead>
<tr>
<th>Processor</th>
<th>Clock</th>
<th>Voltage</th>
<th>Dynamic P</th>
<th>Static P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pentium 4</td>
<td>3.6 GHz</td>
<td>1.25 V</td>
<td>90 W</td>
<td>10 W</td>
</tr>
<tr>
<td>Core i5</td>
<td>3.4 GHz</td>
<td>0.9 V</td>
<td>40 W</td>
<td>30 W</td>
</tr>
</tbody>
</table>

Calculate capacitive load of each processor.

If total power is to be reduced by 10%, how much should the voltage be reduced?
Example

- For given processor, assume we reduce the voltage by 10% and increase the frequency by 5%. What is the improvement to dynamic power consumption?

\[
\frac{\text{power}_{\text{orig}}}{\text{power}_{\text{new}}} = \frac{CV^2F}{C(0.9V)^2(1.05)F} = \frac{V^2}{(0.9V)^2(1.05)}
\]

\[
= \frac{V^2}{0.81V^2(1.05)} = \frac{1}{0.81(1.05)} = 1.18
\]
Fallacy: Low Power at Idle

- **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
SPEC Power Benchmark

- Power consumption of server at different workload levels
  - Performance: ssj_ops/sec
    - ssj_ops = server side Java operations per second
  - Power: Watts (Joules/sec)

\[
\text{Overall ssj\_ops\_per\_Watt} = \frac{\left( \sum_{i=0}^{10} \text{ssj\_ops}_i \right)}{\left( \sum_{i=0}^{10} \text{power}_i \right)}
\]
### 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>865,618</td>
<td>258</td>
</tr>
<tr>
<td>90%</td>
<td>786,688</td>
<td>242</td>
</tr>
<tr>
<td>80%</td>
<td>698,051</td>
<td>224</td>
</tr>
<tr>
<td>70%</td>
<td>607,826</td>
<td>204</td>
</tr>
<tr>
<td>60%</td>
<td>521,391</td>
<td>185</td>
</tr>
<tr>
<td>50%</td>
<td>436,757</td>
<td>170</td>
</tr>
<tr>
<td>40%</td>
<td>345,919</td>
<td>157</td>
</tr>
<tr>
<td>30%</td>
<td>262,071</td>
<td>146</td>
</tr>
<tr>
<td>20%</td>
<td>176,061</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>4,787,166</td>
<td>1,922</td>
</tr>
</tbody>
</table>

\[
\frac{\sum_{ssj\_ops}}{\sum_{power}} = 2,490
\]
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 Intel Core i7 920

<table>
<thead>
<tr>
<th>Description</th>
<th>Name</th>
<th>Instruction Count $\times 10^8$</th>
<th>CPI</th>
<th>Clock cycle time (seconds $\times 10^{-9}$)</th>
<th>Executive Time (seconds)</th>
<th>Reference Time (seconds)</th>
<th>SPECratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interpreted string processing</td>
<td>perl</td>
<td>2,252</td>
<td>0.60</td>
<td>0.376</td>
<td>508</td>
<td>9,770</td>
<td>19.2</td>
</tr>
<tr>
<td>Block-sorting compression</td>
<td>bzip2</td>
<td>2,390</td>
<td>0.70</td>
<td>0.376</td>
<td>629</td>
<td>9,650</td>
<td>15.4</td>
</tr>
<tr>
<td>GNU C compiler</td>
<td>gcc</td>
<td>794</td>
<td>1.20</td>
<td>0.376</td>
<td>358</td>
<td>8,050</td>
<td>22.5</td>
</tr>
<tr>
<td>Combinatorial optimization</td>
<td>mcf</td>
<td>221</td>
<td>2.66</td>
<td>0.376</td>
<td>221</td>
<td>9,120</td>
<td>41.2</td>
</tr>
<tr>
<td>Go game (AI)</td>
<td>go</td>
<td>1,274</td>
<td>1.10</td>
<td>0.376</td>
<td>527</td>
<td>10,490</td>
<td>19.9</td>
</tr>
<tr>
<td>Search gene sequence</td>
<td>hmmer</td>
<td>2,616</td>
<td>0.60</td>
<td>0.376</td>
<td>590</td>
<td>9,330</td>
<td>15.8</td>
</tr>
<tr>
<td>Chess game (AI)</td>
<td>sjeng</td>
<td>1,948</td>
<td>0.80</td>
<td>0.376</td>
<td>586</td>
<td>12,100</td>
<td>20.7</td>
</tr>
<tr>
<td>Quantum computer simulation</td>
<td>libquantum</td>
<td>659</td>
<td>0.44</td>
<td>0.376</td>
<td>109</td>
<td>20,720</td>
<td>190.0</td>
</tr>
<tr>
<td>Video compression</td>
<td>h264avc</td>
<td>3,793</td>
<td>0.50</td>
<td>0.376</td>
<td>713</td>
<td>22,130</td>
<td>31.0</td>
</tr>
<tr>
<td>Discrete event simulation library</td>
<td>omnetpp</td>
<td>367</td>
<td>2.10</td>
<td>0.376</td>
<td>290</td>
<td>6,250</td>
<td>21.5</td>
</tr>
<tr>
<td>Games/path finding</td>
<td>astart</td>
<td>1,250</td>
<td>1.00</td>
<td>0.376</td>
<td>470</td>
<td>7,020</td>
<td>14.9</td>
</tr>
<tr>
<td>XML parsing</td>
<td>xalancbmk</td>
<td>1,045</td>
<td>0.70</td>
<td>0.376</td>
<td>275</td>
<td>6,900</td>
<td>25.1</td>
</tr>
<tr>
<td>Geometric Mean</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>25.7</td>
</tr>
</tbody>
</table>
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
Example

- Use Amdahl’s Law to compute the new execution time for an architecture that previously required 25 seconds to execute a program, where 15% of the time was spent executing load/store instructions, if the time required for a load/store operation is reduced by 40% (amount of improvement for load/stores = 1/.60 = 1.67).
Example

• Suppose you have a machine which executes a program consisting of 50% multiply instructions, 20% divide instructions, and the remaining 30% are other instructions. Management wants the machine to run 4 times faster. You can make the divide run at most 3 times faster and the multiply run at most 8 times faster. Can you meet management’s goal by making only one improvement, and which one?
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
Example

- Assume the following instruction classes and corresponding CPIs and dynamic execution counts:

<table>
<thead>
<tr>
<th>Type</th>
<th>CPI</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>arithmetic</td>
<td>A</td>
<td>X</td>
</tr>
<tr>
<td>load/store</td>
<td>B</td>
<td>Y</td>
</tr>
<tr>
<td>branch</td>
<td>C</td>
<td>Z</td>
</tr>
</tbody>
</table>

When run on > 1 processors, the number of executed **arithmetic instructions** is divided by 0.7$p$ (where $p$ = number of processors) but the number of other instructions executed remains the same. To what factor would the CPI of load/store instructions need to be reduced (sped up) in order for a **single processor** to match the performance of **four processors** each having the original CPI?

\[
AX + \frac{BY}{f} + CZ = \frac{AX}{0.7 \cdot 4} + BY + CZ
\]
Example

- Assume the following instruction classes and corresponding CPIs and dynamic execution counts:

<table>
<thead>
<tr>
<th>Type</th>
<th>CPI</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>arithmetic</td>
<td>A</td>
<td>X</td>
</tr>
<tr>
<td>load/store</td>
<td>B</td>
<td>Y</td>
</tr>
<tr>
<td>branch</td>
<td>C</td>
<td>Z</td>
</tr>
</tbody>
</table>

As compared to a single processor, under what condition is it possible to achieve an overall speedup of 6 by using multiple processors? Express this as an inequality.

\[
\frac{1}{6} \geq \frac{A}{A + B + C} \times \frac{.7p}{A + B + C} + \frac{B + C}{A + B + C}
\]
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