Lecture 02: Technology Trends and Quantitative Design and Analysis for Performance

CSE 564 Computer Architecture Summer 2017

Department of Computer Science and Engineering
Yonghong Yan
yan@oakland.edu
www.secs.oakland.edu/~yan
Contents

• Computers and computer components
• Computer architectures and great ideas in history and now
• Trends, Cost and 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
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
Trends in Technology

• Integrated circuit technology
  – Transistor density: 35%/year
  – Die size: 10-20%/year
  – Integration overall: 40-55%/year

• DRAM capacity: 25-40%/year (slowing)

• Flash capacity: 50-60%/year
  – 15-20X cheaper/bit than DRAM

• Magnetic disk technology: 40%/year
  – 15-25X cheaper/bit than Flash
  – 300-500X cheaper/bit than DRAM
Bandwidth and Latency

• Bandwidth or throughput
  – Total work done in a given time
  – 10,000-25,000X improvement for processors
  – 300-1200X improvement for memory and disks

• Latency or response time
  – Time between start and completion of an event
  – 30-80X improvement for processors
  – 6-8X improvement for memory and disks
End\tof\Moore\'\st\Law?\n
Cost\per\transistor\ris\trising\as\transistor\size\continues\to\shrink.

**Shrinking chips**
Number and length of transistors bought per $
Power and Energy

• Problem:
  – Get power in and distribute around
  – Get power out: dissipate heat

• Three primary concerns:
  – Max power requirement for a processor
  – Thermal Design Power (TDP)
    • Characterizes sustained power consumption
    • Used as target for power supply and cooling system
    • Lower than peak power, higher than average power consumption
  – Energy and energy efficiency

• Clock rate can be reduced dynamically to limit power consumption
Energy and Energy Efficiency

• Power: energy per unit time
  – 1 watt = 1 joule per second
  – Energy per task is often a better measurement

• Processor A has 20% higher average power consumption than processor B. A executes task in only 70% of the time needed by B.
  – So energy consumption of A will be $1.2 \times 0.7 = 0.84$ of B
Dynamic Energy and Power

- Dynamic energy
  - Transistor switch from 0 -> 1 or 1 -> 0

\[
\text{Energy}_{\text{dynamic}} \propto \frac{1}{2} \times \text{Capacitive load} \times \text{Voltage}^2
\]

- Dynamic power

\[
\text{Power}_{\text{dynamic}} \propto \frac{1}{2} \times \text{Capacitive load} \times \text{Voltage}^2 \times \text{Frequency switched}
\]

- Reducing clock rate reduces power, not energy
- The capacitive load:
  - a function of the number of transistors connected to an output and the technology, which determines the capacitance of the wires and the transistors.
An Example from Textbook page #21

Some microprocessors today are designed to have adjustable voltage, so a 15% reduction in voltage may result in a 15% reduction in frequency. What would be the impact on dynamic energy and on dynamic power?

Since the capacitance is unchanged, the answer for energy is the ratio of the voltages since the capacitance is unchanged:

\[
\frac{\text{Energy}_{\text{new}}}{\text{Energy}_{\text{old}}} = \frac{(\text{Voltage} \times 0.85)^2}{\text{Voltage}^2} = 0.85^2 = 0.72
\]

thereby reducing energy to about 72% of the original. For power, we add the ratio of the frequencies

\[
\frac{\text{Power}_{\text{new}}}{\text{Power}_{\text{old}}} = 0.72 \times \frac{(\text{Frequency switched} \times 0.85)}{\text{Frequency switched}} = 0.61
\]

shrinking power to about 61% of the original.
An Example from Textbook

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

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

Power = Capacitive load × Voltage^2 × Frequency

×30
5V → 1V
×1000
Power

- Intel 80386 consumed ~ 2 W
- 3.3 GHz Intel Core i7 consumes 130 W
- Heat must be dissipated from 1.5 x 1.5 cm chip
- This is the limit of what can be cooled by air
The Power Wall

• We can’t reduce voltage further
• We can’t remove more heat

• Techniques for reducing power:
  – Do nothing well
    • Turn off clock of inactive module
  – Dynamic Voltage-Frequency Scaling
  – Low power state for DRAM, disks
  – Overclocking, turning off cores
Static Power

• Because of leakage current flows even a transistor is off

  \[
  \text{Power}_{\text{static}} \propto \text{Current}_{\text{static}} \times \text{Voltage}
  \]

• Scales with number of transistors

• Leakage can be as high as 50% for
  – In part because of large SRAM caches

• To reduce: power gating
  – Turn off power of inactive modules
Measuring Performance

- Typical performance metrics:
  - Response time
  - Throughput

- Speedup of X relative to Y
  - Execution time$_Y$ / Execution time$_X$

- Execution time
  - Wall clock time: includes all system overheads
  - CPU time: only computation time

- Benchmarks
  - Kernels (e.g. matrix multiply)
  - Toy programs (e.g. sorting)
  - Synthetic benchmarks (e.g. Dhrystone)
  - Benchmark suites (e.g. SPEC06fp, TPC-C)
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: Speedup

- 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
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
  - “time” command in Linux
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
  - Clock period: 1/(4.0×10^9) s = 0.25ns
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 $1.2 \times \text{clock cycles of A}$
• 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

Clock Cycles = Instruction Count × Cycles per Instruction

CPU Time = Instruction Count × CPI × Clock Cycle Time

= Instruction Count × CPI

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?

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

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
    = 2×1 + 1×2 + 2×3
    = 10
  - Avg. CPI = 10/5 = 2.0

- Sequence #2: IC = 6
  - Clock Cycles
    = 4×1 + 1×2 + 1×3
    = 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, $T_c$
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)
Principles of Computer Design

• The Processor Performance Equation

\[
\text{CPU time} = \text{CPU clock cycles for a program} \times \text{Clock cycle time}
\]

\[
\text{CPU time} = \frac{\text{CPU clock cycles for a program}}{\text{Clock rate}}
\]

\[
\text{CPI} = \frac{\text{CPU clock cycles for a program}}{\text{Instruction count}}
\]

\[
\text{CPU time} = \text{Instruction count} \times \text{Cycles per instruction} \times \text{Clock cycle time}
\]

\[
\frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock cycle}} = \frac{\text{Seconds}}{\text{Program}} = \text{CPU time}
\]
Principles of Computer Design

• Different instruction types having different CPIs

\[
\text{CPU clock cycles} = \sum_{i=1}^{n} \text{IC}_i \times \text{CPI}_i
\]

\[
\text{CPU time} = \left( \sum_{i=1}^{n} \text{IC}_i \times \text{CPI}_i \right) \times \text{Clock cycle time}
\]
Metrics of Performance

Application

Programming Language

Compiler

Datapath

Function Units

Control

Cycles per second (clock rate)

Transistors

Wires

Pins

(millions) of Instructions per second: MIPS

(millions) of (FP) operations per second: MFLOP/s

Megabytes per second

Answers per day/month
**Impacts by Components**

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

<table>
<thead>
<tr>
<th></th>
<th>Inst Count</th>
<th>CPI</th>
<th>Clock Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compiler</td>
<td>X</td>
<td>(X)</td>
<td></td>
</tr>
<tr>
<td>Inst. Set.</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Architecture</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Technology</td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>
Principles of Computer Design

• Take Advantage of Parallelism
  – e.g. multiple processors, disks, memory banks, pipelining, multiple functional units

• Principle of Locality
  – Reuse of data and instructions

• Focus on the Common Case
  – Amdahl’s Law

\[
\begin{align*}
\text{Execution time}_{\text{new}} &= \text{Execution time}_{\text{old}} \times \left(1 - \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}\right) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}} \\
\text{Speedup}_{\text{overall}} &= \frac{\text{Execution time}_{\text{old}}}{\text{Execution time}_{\text{new}}} = \frac{1}{(1 - \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}}
\end{align*}
\]
Amdahl’s Law

\[ \text{ExTime}_{\text{new}} = \text{ExTime}_{\text{old}} \times \left[ (1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}} \right] \]

\[ \text{Speedup}_{\text{overall}} = \frac{\text{ExTime}_{\text{old}}}{\text{ExTime}_{\text{new}}} = \frac{1}{(1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}} \]

Best you could ever hope to do:

\[ \text{Speedup}_{\text{maximum}} = \frac{1}{(1 - \text{Fraction}_{\text{enhanced}})} \]
Using Amdahl’s Law

Overall speedup if we make 90% of a program run 10 times faster.

\[
F = 0.9 \quad S = 10
\]

\[
\text{Overall Speedup} = \frac{1}{(1 - 0.9) + \frac{0.9}{10}} = \frac{1}{0.1 + 0.09} = 5.26
\]

Overall speedup if we make 80% of a program run 20% faster.

\[
F = 0.8 \quad S = 1.2
\]

\[
\text{Overall Speedup} = \frac{1}{(1 - 0.8) + \frac{0.8}{1.2}} = \frac{1}{0.2 + 0.66} = 1.153
\]
Amdahl’s Law for Parallelism

- The enhanced fraction $F$ is through parallelism, perfect parallelism with linear speedup
  - The speedup for $F$ is $N$ for $N$ processors
- Overall speedup

$$S(N) = \frac{T_s}{T_p} = \frac{T_s}{(1-F)*T_s + \frac{F*T_s}{N}} = \frac{1}{1-F + \frac{F}{N}}$$

- Speedup upper bound (when $N \to \infty$):
  - $1-F$: the sequential portion of a program
Amdahl’s Law for Parallelism
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
Exercise #1: Amdahl’s Law

Suppose that we want to enhance the processor used for Web serving. The new processor is 10 times faster on computation in the Web serving application than the original processor. Assuming that the original processor is busy with computation 40% of the time and is waiting for I/O 60% of the time, what is the overall speedup gained by incorporating the enhancement?
Exercise #1: Amdahl’s Law solution

- Textbook page #47

\[ \text{Fraction}_{\text{enhanced}} = 0.4; \quad \text{Speedup}_{\text{enhanced}} = 10; \]

\[ \text{Speedup}_{\text{overall}} = \frac{1}{0.6 + \frac{0.4}{10}} = \frac{1}{0.64} \approx 1.56 \]
Exercise #2: CPU time and speedup

Suppose we have made the following measurements:

- Frequency of FP operations = 25%
- Average CPI of FP operations = 4.0
- Average CPI of other instructions = 1.33
- Frequency of FPSQR = 2%
- CPI of FPSQR = 20

Assume that the two design alternatives are to decrease the CPI of FPSQR to 2 or to decrease the average CPI of all FP operations to 2.5. Compare these two design alternatives using the processor performance equation.
Exercise #2: solution, textbook page 51

First, observe that only the CPI changes; the clock rate and instruction count remain identical. We start by finding the original CPI with neither enhancement:

\[
\text{CPI}_{\text{original}} = \sum_{i=1}^{n} \text{CPI}_i \times \left( \frac{\text{IC}_i}{\text{Instruction count}} \right) \\
= (4 \times 25\%) + (1.33 \times 75\%) = 2.0
\]

We can compute the CPI for the enhanced FPSQR by subtracting the cycles saved from the original CPI:

\[
\text{CPI}_{\text{with new FPSQR}} = \text{CPI}_{\text{original}} - 2\% \times (\text{CPI}_{\text{old FPSQR}} - \text{CPI}_{\text{of new FPSQR only}}) \\
= 2.0 - 2\% \times (20 - 2) = 1.64
\]

We can compute the CPI for the enhancement of all FP instructions the same way or by summing the FP and non-FP CPIs. Using the latter gives us:

\[
\text{CPI}_{\text{new FP}} = (75\% \times 1.33) + (25\% \times 2.5) = 1.625
\]

Since the CPI of the overall FP enhancement is slightly lower, its performance will be marginally better. Specifically, the speedup for the overall FP enhancement is

\[
\text{Speedup}_{\text{new FP}} = \frac{\text{CPU time}_{\text{original}}}{\text{CPU time}_{\text{new FP}}} = \frac{\text{IC} \times \text{Clock cycle} \times \text{CPI}_{\text{original}}}{\text{IC} \times \text{Clock cycle} \times \text{CPI}_{\text{new FP}}} \\
= \frac{\text{CPI}_{\text{original}}}{\text{CPI}_{\text{new FP}}} = \frac{2.00}{1.625} = 1.23
\]