Chapter 1: Fundamentals of Computer Design (Part 2)

What is computer architecture?
Why study computer architecture?

Common principles

Performance
  What is performance: latency, throughput
  The performance equation
  Measuring performance
  Improving performance: parallelism, locality, Amdahl's law

Power

Cost

Reliability
What is Performance?

Two Metrics

- Latency (or response time or execution time)
- Throughput (or bandwidth)
What is Performance?**

Two Metrics

- **Latency (or response time or execution time)**
  - Time from start to finish of a task
- **Throughput (or bandwidth)**
What is Performance?**

Two Metrics

- Latency (or response time or execution time)
  - Time from start to finish of a task
- Throughput (or bandwidth)
  - Rate of task completion
    - = Rate of task initiation
    - = 1 / (time between task completions)
What is Performance?**

Two Metrics

Latency (or response time or execution time)
Time from start to finish of a task

Throughput (or bandwidth)
Rate of task completion
= Rate of task initiation
= 1 / (time between task completions)

Example: Automobile assembly line starts one car per hour and holds 20 cars
Latency = 20 hours
Throughput = one car per hour
Throughput > 1/Latency due to overlap
Definition: X is n% faster than Y if

\[
\frac{\text{Execution Time}_Y}{\text{Execution Time}_X} = 1 + \frac{n}{100}
\]

Example: X = 1 minute, Y = 2 minutes
X is 100% faster than Y
**Key Performance Equation**

\[ CPU_{time} = \frac{\text{instructions}}{\text{program}} \times \frac{\text{cycles}}{\text{instruction}} \times \frac{\text{time}}{\text{cycle}} \]

Instructions per program (path length)
ISA and compiler

Cycles per instruction (CPI)
ISA and organization (e.g., cache misses)

Time per cycle (clock time, cycle time)
Organization and hardware
Measuring Performance

MIPS, MFLOPS don't mean much

Benchmarks
  - Real programs
    - Representative of real workload
    - Only way to characterize performance
  - SPEC89 → SPEC92 → SPEC95 → SPEC CPU2000 → CPU2006 → CPU2017
  - SPECFS, SPECWeb, SPECjbb, SPECvirt_Sc2010, TPC

Kernels
  - "Representative" program fragments
  - Often not representative of full applications
  - EEMBC for embedded systems

Toy benchmarks and synthetic benchmarks
  - Don't mean much
Improving Performance – Basic Principles

Parallelism

Locality

Focus on common case – Amdahl’s law
Improving Performance – Basic Principles**

Parallelism
- Pipelining (review next)
- Multiple issue
- Multiprocessors

Locality

Focus on common case – Amdahl’s law
Improving Performance – Basic Principles**

Parallelism

- Pipelining (review next)
- Multiple issue
- Multiprocessors

Locality

- Caches (review later)

Focus on common case – Amdahl’s law
Amdahl's Law

(Or why the common case matters most)

Let

\[ \text{Speedup} = \frac{\text{new rate}}{\text{old rate}} = \frac{\text{old latency}}{\text{new latency}} \]

Consider an enhancement \( x \) that speeds up fraction \( f_x \) of a task by \( S_x \)

\[ \text{Speedup}_{\text{overall}} = \frac{\text{old latency}}{\text{new latency}} \]
\[ = \frac{\{(1 - f_x) + (f_x)\} \times \text{old latency}}{(1 - f_x) \times \text{old latency} + f_x / S_x \times \text{old latency}} \]

Amdahl’s law gives

\[ \text{Speedup}_{\text{overall}} = \frac{1}{(1 - f_x) + f_x / S_x} \]
Amdahl's Law, cont.

Example: \( f_x = 95\% \) and \( S_x = 1.10 \)

\[
\text{Speedup}_{\text{overall}} = \frac{1}{(1 - 0.95) + (0.95/1.10)} = 1.094
\]

Example: \( f_x = 5\% \) and \( S_x = 10 \)

\[
\text{Speedup}_{\text{overall}} = \frac{1}{(1 - 0.05) + (0.05/10)} = 1.047
\]

Example: \( f_x = 5\% \) and \( S_x = \infty \)

\[
\text{Speedup}_{\text{overall}} = \frac{1}{(1 - 0.05) + (0.05/\infty)} = 1.052
\]
Amdahl's Law Corollary

Since $S_x \to \infty$ implies

$$Speedup_{overall} = \frac{1}{(1 - f_x) + \frac{f_x}{\infty}}$$

For all real speedups:

$$Speedup_{overall} < \frac{1}{1 - f_x}$$

Or make the common case fast

An application?

<table>
<thead>
<tr>
<th>$f_x$</th>
<th>$1/(1-f_x)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1%</td>
<td>1.01</td>
</tr>
<tr>
<td>2%</td>
<td>1.02</td>
</tr>
<tr>
<td>5%</td>
<td>1.05</td>
</tr>
<tr>
<td>10%</td>
<td>1.11</td>
</tr>
<tr>
<td>20%</td>
<td>1.25</td>
</tr>
<tr>
<td>50%</td>
<td>2.00</td>
</tr>
</tbody>
</table>
Power

Energy

Temperature
**Power**

Power = Voltage x Current

Energy = Power x Time

Temperature = Complex function of power, determined by max power drawn (averaged over short intervals)
Power and Energy

Power = Dynamic power + Static power
Energy = Power * Time
Dynamic Power $\propto$ Capacitance * Voltage$^2$ * Frequency

Static power = Static current * Voltage
**Power and Energy**

Power = Dynamic power + Static power

Energy = Power * Time

Dynamic Power $\propto$ Capacitance * Voltage$^2$ * Frequency

- Clock gating reduces switching to reduce power
- Reducing voltage reduces power
  - But also requires reducing frequency
  - Increases execution time
  - But reduction in power (and energy) more than increase in time

*Dynamic voltage-frequency scaling (DVFS)*

Static power = Static current * Voltage

- Power gating reduces static (and dynamic) power

System strategy for energy: Race-to-halt

- New reality: Dark Silicon
Figure 1.11 Growth in clock rate of microprocessors in Figure 1.1. Between 1978 and 1986, the clock rate improved less than 15% per year while performance improved by 22% per year. During the “renaissance period” of 52% performance improvement per year between 1986 and 2003, clock rates shot up almost 40% per year. Since then, the clock rate has been nearly flat, growing at less than 2% per year, while single processor performance improved recently at just 3.5% per year.

© 2019 Elsevier Inc. All rights reserved.
Cost is very important in most real designs
   But usually hard to quantify for the architect

Costs change over time
   Learning curve lowers manufacturing costs
   Technology improvements lower costs

Focus on IC costs next
Figure 1.15 This 300 mm wafer contains 280 full Sandy Bridge dies, each 20.7 by 10.5 mm in a 32 nm process. (Sandy Bridge is Intel’s successor to Nehalem used in the Core i7.) At 216 mm2, the formula for dies per wafer estimates 282. (Courtesy Intel.)
Figure 1.14 Photograph of an Intel Skylake microprocessor die, which is evaluated in Chapter 4.
Figure 1.15 The components of the microprocessor die in Figure 1.14 are labeled with their functions.
**Integrated Circuit Cost**

Cost of IC = \( \frac{\text{Cost of Die} + \text{Cost of Testing} + \text{Cost of Packaging}}{\text{Final Test Yield}} \)

Cost of Die = \( \frac{\text{Cost of Wafer}}{\text{Dies per Wafer} \times \text{Die Yield}} \)

Dies per Wafer = \( (\frac{\pi \times (\text{Wafer Diameter}/2)^2}{\text{Die Area}}) - \) \( (\text{Correction factor for Edge Effects}) \)

Die Yield = Wafer Yield \( \times \) \( \frac{1}{(1 + \text{Defects per unit area} \times \text{Die Area})^\alpha} \)

\( \alpha = 10 \) to \( 14 \) for \( 16\text{nm} \) in \( 2017 \)

Bottom line: Cost per die grows roughly as the square of the die area
Cost different from price; cost of manufacturing different from cost of operation
Many sources of unreliability

- Soft errors due to radiation, hard errors due to wearout, …

Common metrics

- Mean time to failure – MTTF

For exponentially distributed time to failure

Define failures in time or FITs

- FIT = failures in a billion hours
- FIT $\alpha \frac{1}{\text{MTTF}}$
- FIT of system = Sum of FITs of components

Common solution
Many sources of unreliability
   Soft errors due to radiation, hard errors due to wearout, ...

Common metrics
   Mean time to failure – MTTF
   For exponentially distributed time to failure
      Define failures in time or FITs
         FIT = failures in a billion hours
         FIT $\alpha 1/\text{MTTF}$
         FIT of system = Sum of FITs of components

Common solution
   Redundancy in time, space, information
   But must be cheap
      Many recent innovations