Lecture 2:
Computer Performance

Assist.Prof.Dr. Gürhan Küçük
Computer Architecture
CSE 427
Performance and Cost

° Purchasing perspective
  • given a collection of machines, which has the
    - best performance ?
    - least cost ?
    - best performance / cost ?

° Design perspective
  • faced with design options, which has the
    - best performance improvement ?
    - least cost ?
    - best performance / cost ?

° Both require
  • basis for comparison
  • metric for evaluation

° Our goal is to understand cost & performance implications of architectural choices
Building Computer Chips: Cost & Timing

° Complex multi-step process
  • Slice ingots into wafers
  • Process wafers into patterned wafers
  • Dice patterned wafers into dies
  • Test dies, select good dies
  • Bond to package
  • Test parts
  • Ship to customers and make money
Building Computer Chips: Cost

1. Silicon ingot
2. Slicer
3. Blank wafers
4. 20 to 30 processing steps
5. Patterned wafers
6. Dicer
7. Individual dies (one wafer)
8. Die tester
9. Tested dies
10. Bond die to package
11. Packaged dies
12. Part tester
13. Tested packaged dies
14. Ship to customers
Integrated Circuits Costs

\[
\text{IC cost} = \text{Die cost} + \text{Testing cost} + \text{Packaging cost} \over \text{Final test yield}
\]

\[
\text{Die cost} = \frac{\text{Wafer cost}}{\text{Dies per Wafer} \times \text{Die yield}}
\]

\[
\text{Dies per wafer} = \frac{\pi \times (\text{Wafer diam} / 2)^2}{\text{Die Area}} - \frac{\pi \times \text{Wafer diam}}{2 \times \text{Die Area}} - \text{Test dies}
\]

\[
\text{Die Yield} = \text{Wafer yield} \times \left\{ 1 + \frac{\text{Defects per unit area} \times \text{Die Area}}{\alpha} \right\}^{-\alpha}
\]

Die Cost goes roughly with die area\(^4\)
## Real World Examples

<table>
<thead>
<tr>
<th>Chip</th>
<th>Metal layers</th>
<th>Line width</th>
<th>Wafer cost</th>
<th>Defect/cm²</th>
<th>Area/mm²</th>
<th>Dies/wafer</th>
<th>Yield</th>
<th>Die Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>386DX</td>
<td>2</td>
<td>0.90</td>
<td>$900</td>
<td>1.0</td>
<td>43</td>
<td>360</td>
<td>71%</td>
<td>$4</td>
</tr>
<tr>
<td>486DX2</td>
<td>3</td>
<td>0.80</td>
<td>$1200</td>
<td>1.0</td>
<td>81</td>
<td>181</td>
<td>54%</td>
<td>$12</td>
</tr>
<tr>
<td>PowerPC 601</td>
<td>4</td>
<td>0.80</td>
<td>$1700</td>
<td>1.3</td>
<td>121</td>
<td>115</td>
<td>28%</td>
<td>$53</td>
</tr>
<tr>
<td>HP PA 7100</td>
<td>3</td>
<td>0.80</td>
<td>$1300</td>
<td>1.0</td>
<td>196</td>
<td>66</td>
<td>27%</td>
<td>$73</td>
</tr>
<tr>
<td>DEC Alpha</td>
<td>3</td>
<td>0.70</td>
<td>$1500</td>
<td>1.2</td>
<td>234</td>
<td>53</td>
<td>19%</td>
<td>$149</td>
</tr>
<tr>
<td>SuperSPARC</td>
<td>3</td>
<td>0.70</td>
<td>$1700</td>
<td>1.6</td>
<td>256</td>
<td>48</td>
<td>13%</td>
<td>$272</td>
</tr>
<tr>
<td>Pentium</td>
<td>3</td>
<td>0.80</td>
<td>$1500</td>
<td>1.5</td>
<td>296</td>
<td>40</td>
<td>9%</td>
<td>$417</td>
</tr>
</tbody>
</table>

Timing is also Important:

Performance vs. Design Time

° Time to market is critically important

° E.g., a new design may take 3 years
  • It will be 3 times faster
  • But if technology improves 50%/year
  • In 3 years $1.5^3 = 3.38$
  • So the new design is worse!
    (unless it also employs new technology)
Two notions of “performance”

<table>
<thead>
<tr>
<th>Plane</th>
<th>DC to Paris</th>
<th>Speed</th>
<th>Passengers</th>
<th>Throughput (pmph)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boeing 747</td>
<td>6.5 hours</td>
<td>610 mph</td>
<td>470</td>
<td>286,700</td>
</tr>
<tr>
<td>BAD/Sud Concorde</td>
<td>3 hours</td>
<td>1350 mph</td>
<td>132</td>
<td>178,200</td>
</tr>
</tbody>
</table>

Which has higher performance?

° **Time to do the task** (Execution Time)
  – execution time, response time, latency

° **Tasks per day, hour, week, sec, ns. ..** (Performance)
  – performance, throughput, bandwidth

Response time and throughput often are in opposition - why?
Performance Definitions

° Performance is in units of things-per-second
  • bigger is better

° If we are primarily concerned with response time
  • performance(x) = \frac{1}{\text{execution_time}(x)}

° "X is n times faster than Y" means
  \[
  \frac{\text{performance}(X)}{\text{execution_time}(Y)} = \frac{\text{performance}(Y)}{\text{execution_time}(X)}
  \]

° When is throughput more important than execution time?
° When is execution time more important than throughput?
Performance Examples

- **Time of Concorde vs. Boeing 747?**
  - Concorde is 1350 mph / 610 mph = 2.2 times faster  
    = 6.5 hours / 3 hours

- **Throughput of Concorde vs. Boeing 747?**
  - Concorde is 178,200 pmph / 286,700 pmph = 0.62 “times faster”
  - Boeing is 286,700 pmph / 178,200 pmph = 1.6 “times faster”

- **Boeing** is 1.6 times (“60%”) faster in terms of throughput  
- **Concorde** is 2.2 times (“120%”) faster in terms of flying time

- When discussing processor performance, we will focus primarily on execution time for a single job - why?
Understanding Performance

How are the following likely to effect response time and throughput?

- Increasing the clock speed of a given processor.

- Increasing the number of jobs in a system (e.g., having a single computer service multiple users).

- Increasing the number of processors in a system that uses multiple processors (e.g., a network of ATM machines).

If a **Pentium III** runs a program in 8 seconds and a **PowerPC** runs the same program in 10 seconds, how many times faster is the Pentium?

\[ n = \frac{10}{8} = 1.25 \text{ times faster (or 25% faster)} \]

Why might someone chose to buy the PowerPC in this case?
Time can be defined in different ways, depending on what we are measuring:

- **Response time**: Total time to complete a task, including time spent executing on the CPU, accessing disk and memory, waiting for I/O and other processes, and operating system overhead.
- **CPU execution time**: Total time a CPU spends computing on a given task (excludes time for I/O or running other programs). This is also referred to as simply **CPU time**.
- **User CPU time**: Total time CPU spends in the program.
- **System CPU execution time**: Total time operating systems spends executing tasks for the program.

For example, a program may have a **system CPU time** of 22 sec., a **user CPU time** of 90 sec., a **CPU execution time** of 112 sec., and a **response time** of 162 sec.
Computer Clocks

- A computer clock runs at a constant rate and determines when events take place in hardware.

- The clock cycle time is the amount of time for one clock period to elapse (e.g. 5 ns).

- The clock rate is the inverse of the clock cycle time.

- For example, if a computer has a clock cycle time of 5 ns, the clock rate is:

\[
\frac{1}{5 \times 10^{-9} \text{ sec}} = 200 \text{ MHz}
\]
Computing CPU time

° The time to execute a given program can be computed as
  \[ \text{CPU time} = \text{CPU clock cycles} \times \text{clock cycle time} \]

Since clock cycle time and clock rate are reciprocals
  \[ \text{CPU time} = \frac{\text{CPU clock cycles}}{\text{clock rate}} \]

° The number of CPU clock cycles can be determined by
  \[ \text{CPU clock cycles} = \left( \frac{\text{instructions}}{\text{program}} \right) \times \left( \frac{\text{clock cycles}}{\text{instruction}} \right) \]
  \[ = \text{Instruction count} \times \text{CPI} \]

which gives
  \[ \text{CPU time} = \text{Instruction count} \times \text{CPI} \times \text{clock cycle time} \]
  \[ = \frac{\text{Instruction count} \times \text{CPI}}{\text{clock rate}} \]

° The units for this are
  \[ \text{seconds} = \frac{\text{instructions}}{\text{program}} \times \frac{\text{clock cycles}}{\text{instruction}} \times \frac{\text{seconds}}{\text{clock cycle}} \]
If a computer has a clock rate of 50 MHz, how long does it take to execute a program with 1,000 instructions, if the CPI for the program is 3.5?

Using the equation

\[
\text{CPU time} = \frac{\text{Instruction count} \times \text{CPI}}{\text{clock rate}}
\]

gives

\[
\text{CPU time} = \frac{1000 \times 3.5}{50 \times 10^6} = 70 \, \mu\text{sec}
\]

If a computer’s clock rate increases from 200 MHz to 250 MHz and the other factors remain the same, how many times faster will the computer be?

\[
\frac{\text{CPU time old}}{\text{clock rate new}} = \frac{\text{CPU time new}}{\text{clock rate old}}
\]

\[
\frac{\text{CPU time old}}{250 \, \text{MHz}} = \frac{\text{CPU time new}}{200 \, \text{MHz}} = 1.25
\]

What simplifying assumptions did we make?
Iron Law in Detail

Processor Performance = \( T_{\text{exec}} \) = \( \frac{\text{Time}}{\text{Program}} \)

\[
\text{Instructions} \quad \text{Program} \quad \times \quad \text{Cycles} \quad \text{Instruction} \quad \times \quad \text{Time} \quad \text{Cycle}
\]

\[
= (\text{code size N}) \quad \times \quad (\text{CPI}) \quad \times \quad (\text{cycle time } \tau)
\]

Architecture --> Implementation --> Realization

Compiler Designer  Processor Designer  Chip Designer
Iron Law in Detail

° Instructions/Program
  • Instructions executed, not static code size (a.k.a. dynamic instruction count)
  • Determined by algorithm, compiler, ISA

° Cycles/Instruction
  • Determined by ISA and CPU organization
  • Overlap among instructions reduces this term

° Time/cycle
  • Determined by technology, organization, clever circuit design
Factors affecting CPU Performance

Which factors are affected by each of the following?

<table>
<thead>
<tr>
<th></th>
<th>instr. 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>Instr. Set Arch.</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Organization</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Technology</td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

CPU time = Seconds = Instructions x Cycles x Seconds

Program Program Instruction Instruction Instruction Instruction Cycle Cycle Cycle
Our Goal

° Minimize time which is the product, NOT isolated terms

° Common error to miss terms while devising optimizations
  • E.g. ISA change to decrease instruction count
  • BUT leads to CPU organization which makes clock slower

° Bottom line: terms are inter-related
RISC vs. CISC

RISCs (Reduced Instruction Set Computers) and CISCs (Complex Instruction Set Computers):

- RISC CPUs try to reduce $T_{exec}$ by:
  - reducing CPI
  - reducing HW complexity so that we can use faster clocks (shorter cycle times)
- CISC CPUs try to reduce $T_{exec}$ by:
  - reducing $N$ (number of instructions executed)
RISC ISA

1. Load/Store Architecture
2. Instruction semantics are at low level
3. Small number of addressing modes
4. Uniform instruction format

1 & 2: Fast operation
2 & 3 & 4: Fetching and decoding of instructions is very quick

Advantages:
- HW is simple
- Logic delays are smaller => we can use faster clock

Disadvantages:
- Because of 2nd item => N goes up

Design Challenge:
- Growth in N should be compensated by the reduction in CPI and cycle time
CISC ISA

$T_{exec} = N \times CPI \times \text{cycle\_time}$

1. Many “operate” instructions with at least one memory operand
2. Instruction semantics are at high level
3. Many addressing modes
4. Non-uniform instruction format

1 & 2 : Slow operation
2 & 3 & 4 : decoding of instructions is very slow

**Advantages:**
- $N$ is smaller

**Disadvantages:**
- Complex HW
- Larger logic delays => faster clock is not possible
- Higher CPI

**Design Challenge:**
- Growth in CPI and cycle\_time should be compensated by the reduction in $N$
Computing CPI

° The CPI is the average number of cycles per instruction.

° If for each instruction type, we know its frequency and number of cycles need to execute it, we can compute the overall CPI as follows:

\[
\text{CPI} = \sum_{i=1}^{n} \text{CPI}_i \times F_i
\]

° For example

<table>
<thead>
<tr>
<th>Op</th>
<th>F_i</th>
<th>CPI_i</th>
<th>CPI_i \times F_i</th>
<th>% Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU</td>
<td>50%</td>
<td>1</td>
<td>.5</td>
<td>23%</td>
</tr>
<tr>
<td>Load</td>
<td>20%</td>
<td>5</td>
<td>1.0</td>
<td>45%</td>
</tr>
<tr>
<td>Store</td>
<td>10%</td>
<td>3</td>
<td>.3</td>
<td>14%</td>
</tr>
<tr>
<td>Branch</td>
<td>20%</td>
<td>2</td>
<td>.4</td>
<td>18%</td>
</tr>
<tr>
<td>Total</td>
<td>100%</td>
<td>2.2</td>
<td></td>
<td>100%</td>
</tr>
</tbody>
</table>
Suppose we have made the following measurements:

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

Assume that the two design alternatives are
1) To reduce the CPI of FPSQR to 2, or
2) To reduce the average CPI of all FP operations to 2.

Compare these two design alternatives using the CPU performance equation (a.k.a. The Iron Law)
Example (pg. 37 in Hennesy & Patterson)

Suppose we are considering two alternatives for our conditional branch instructions, as follows:

- CPU A: A condition code is set by a compare instruction and followed by a branch that tests the condition code
- CPU B: A compare is included in the branch

On both CPUs, the conditional branch instruction takes 2 cycles, and all other instructions take 1 clock cycle.

On CPU A, 20% of all instructions executed are conditional branches; since every branch needs a compare, another 20% of the instructions are compares.

Because CPU A does not have the compare included in the branch, assume that its clock cycle time is 1.25 times faster (less) than that of CPU B.

Which CPU is faster?
The two main measure of performance are

- **execution time**: time to do the task
- **throughput**: number of tasks completed per unit time

Performance and execution time are reciprocals. Increasing performance, decreases execution time.

The time to execute a given program can be computed as:

\[ \text{CPU time} = \text{Instruction count} \times \text{CPI} \times \text{clock cycle time} \]
\[ \text{CPU time} = \text{Instruction count} \times \text{CPI} / \text{clock rate} \]

These factors are affected by compiler technology, the instruction set architecture, the machine organization, and the underlying technology.

When trying to improve performance, look at what occurs frequently => make the common case fast.
Computer Benchmarks

° A benchmark is a program or set of programs used to evaluate computer performance.

° Benchmarks allow us to make performance comparisons based on execution times

° Benchmarks should
  • Be representative of the type of applications run on the computer
  • Not be overly dependent on one or two features of a computer

° Benchmarks can vary greatly in terms of their complexity and their usefulness.
Types of Benchmarks

**Pros**
- representative
- portable
- widely used
- improvements useful in reality
- easy to run, early in design cycle
- identify peak capability and potential bottlenecks

**Actual Target Workload**
- Full Application Benchmarks (e.g., SPEC benchmarks)
- Small “Kernel” Benchmarks
- Microbenchmarks

**Cons**
- very specific
- non-portable
- difficult to run, or measure
- hard to identify cause
- less representative
- does not measure memory system
- “peak” may be a long way from application performance
SPEC: System Performance Evaluation Cooperative

° The SPEC Benchmarks are the most widely used benchmarks for reporting workstation and PC performance.

• First Round SPEC CPU89
  - 10 programs yielding a single number

• Second Round SPEC CPU92
  - SPEC CINT92 (6 integer programs) and SPEC CFP92 (14 floating point programs)
  - Compiler flags can be set differently for different programs

• Third Round SPEC CPU95
  - New set of programs: SPEC CINT95 (8 integer programs) and SPEC CFP95 (10 floating point)
  - Single compiler flag setting for all programs

• Fourth Round SPEC CPU2000
  - New set of programs: SPEC CINT2000 (12 integer programs) and SPEC CFP2000 (14 floating point)
  - Single compiler flag setting for all programs

° Value reported is the SPEC ratio
  ° CPU time of reference machine / CPU time of measured machine
Other SPEC Benchmarks

° JVM98:
  • Measures performance of Java Virtual Machines

° SFS97:
  • Measures performance of network file server (NFS) protocols

° Web99:
  • Measures performance of World Wide Web applications

° HPC96:
  • Measures performance of large, industrial applications

° APC, MEDIA, OPC
  • Measures performance of graphics applications

° For more information about the SPEC benchmarks see: http://www.spec.org.
Examples of SPEC95 Benchmarks

° SPEC ratios are shown for the Pentium and the Pentium Pro (Pentium+) processors

<table>
<thead>
<tr>
<th>Clock Rate</th>
<th>Pentium SPECint</th>
<th>Pentium+ SPECint</th>
<th>Pentium SPECfp</th>
<th>Pentium+ SPECfp</th>
</tr>
</thead>
<tbody>
<tr>
<td>100 MHz</td>
<td>3.2</td>
<td>N/A</td>
<td>2.6</td>
<td>N/A</td>
</tr>
<tr>
<td>150 MHZ</td>
<td>4.4</td>
<td>6.0</td>
<td>3.0</td>
<td>5.1</td>
</tr>
<tr>
<td>200 MHZ</td>
<td>5.5</td>
<td>8.0</td>
<td>3.8</td>
<td>6.8</td>
</tr>
</tbody>
</table>

° What can we learn from this information?
Marketing metrics for computer performance included MIPS and MFLOPS.

**MIPS**: millions of instructions per second
- MIPS = instruction count / (execution time x $10^6$) =
- MIPS = Clock Rate / (CPI x $10^6$)
- For example, a program that executes 3 million instructions in 2 seconds has a MIPS rating of 1.5
- Advantage: Easy to understand and measure
- Disadvantages: May not reflect actual performance, since simple instructions do better.

**MFLOPS**: millions of floating point operations per second
- MFLOPS = floating point operations / (execution time x $10^6$)
- For example, a program that executes 4 million instructions in 5 seconds has a MFLOPS rating of 0.8
- Advantage: Easy to understand and measure
- Disadvantages: Same as MIPS, only measures floating point operations.
Example (pg. 45 in Hennesy & Patterson)

° Assume we build an optimizing compiler for the load-store machine for which the measurements in Figure below have been made

° The compiler discards 50% of the arithmetic logic unit (ALU) instructions, although it cannot reduce loads, stores, or branches

° Ignoring systems issues and assuming a 2 ns. clock cycle time and 1.57 unoptimized CPI,

° What is the MIPS rating for optimized code versus unoptimized code?

° Does the ranking of MIPS agree with the ranking of execution time?

<table>
<thead>
<tr>
<th>Instruction Type</th>
<th>Frequency</th>
<th>Clock Cycle Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU ops</td>
<td>43%</td>
<td>1</td>
</tr>
<tr>
<td>Loads</td>
<td>21%</td>
<td>2</td>
</tr>
<tr>
<td>Stores</td>
<td>12%</td>
<td>2</td>
</tr>
<tr>
<td>Branches</td>
<td>24%</td>
<td>2</td>
</tr>
</tbody>
</table>
Amdahl's Law

Speedup due to an enhancement is defined as:

\[
\text{Speedup} = \frac{\text{ExTime}_{\text{old}}}{\text{ExTime}_{\text{new}}} = \frac{\text{Performance}_{\text{new}}}{\text{Performance}_{\text{old}}} \]

Suppose that an enhancement accelerates a fraction \(\text{Fraction}_{\text{enhanced}}\) of the task by a factor \(\text{Speedup}_{\text{enhanced}}\),

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

\[
\text{Speedup} = \frac{\text{ExTime}_{\text{old}}}{\text{ExTime}_{\text{new}}} = \frac{1}{(1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}} \]
Example of Amdahl’s Law

° Floating point instructions are improved to run twice as fast, but only 10% of the time was spent on these instructions originally. How much faster is the new machine?

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

\[
\text{Speedup} = \frac{1}{(1 - 0.1) + 0.1/2} = 1.053
\]

° The new machine is 1.053 times as fast, or 5.3% faster.

° How much faster would the new machine be if floating point instructions become 100 times faster?

\[
\text{Speedup} = \frac{1}{(1 - 0.1) + 0.1/100} = 1.109
\]
Suppose a cache is 10 times faster than main memory, and suppose that the cache can be used 90% of the time. How much speedup do we gain by using the cache?
Example (pg. 43 in Hennesy & Patterson)

° Assume we have a machine where the CPI is 2.0 when all memory accesses hit in the cache.

° The only data accesses are loads and stores, and these total 40% of the instructions

° If the miss penalty is 25 clock cycles and the miss rate is 2%,

° How much faster would the machine be if all instructions were cache hits?
Estimating Performance Improvements

° Assume a processor currently requires 10 seconds to execute a program and processor performance improves by 50 percent per year.

° By what factor does processor performance improve in 5 years?

\[(1 + 0.5)^5 = 7.59\]

° How long will it take a processor to execute the program after 5 years?

\[\text{ExTime}_{\text{new}} = \frac{10}{7.59} = 1.32 \text{ seconds}\]

° What assumptions are made in the above problem?
Performance Example

° Computers M1 and M2 are two implementations of the same instruction set.

° M1 has a clock rate of 50 MHz and M2 has a clock rate of 75 MHz.

° M1 has a CPI of 2.8 and M2 has a CPI of 3.2 for a given program.

° How many times faster is M2 than M1 for this program?

\[
\begin{align*}
\frac{\text{ExTime}_{M1}}{\text{ExTime}_{M2}} &= \frac{\text{IC}_{M1} \times \text{CPI}_{M1}}{\text{Clock Rate}_{M1}} \quad \frac{\text{IC}_{M2} \times \text{CPI}_{M2}}{\text{Clock Rate}_{M2}} \\
&= \frac{2.8/50}{3.2/75} = \frac{2.8}{50} \times \frac{75}{3.2} = 1.31
\end{align*}
\]

° What would the clock rate of M1 have to be for them to have the same execution time?
Summary of Performance Evaluation

° **Good** benchmarks, such as the SPEC benchmarks, can provide an accurate method for evaluating and comparing computer performance.

° **MIPS** and **MFLOPS** are easy to use, but inaccurate indicators of performance.

° Amdahl’s law provides an efficient method for determining speedup due to an enhancement.

° Make the common case fast!