Overview

• Introduction to Performance
• Aspects of Performance
  ♦ Execution time, Elapsed time, user CPU time
  ♦ CPI, MIPS and MFLOPS
  ♦ Benchmarks
  ♦ Performance Metrics
• Amdahl’s Law

Part of Chapter 1 of the text (4th Edition)
Understanding Computer 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.
Computer Performance

- Why some hardware is better than others for different programs?
- Which factors of system performance are hardware related?
- How does the machine's instruction set affect performance?

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?

➢ Our goal is to understand cost/performance implications of architectural choices
Performance

Consider the following planes

<table>
<thead>
<tr>
<th>Airplane</th>
<th>Passengers</th>
<th>Range (mi)</th>
<th>Speed (mph)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boeing 777</td>
<td>375</td>
<td>4630</td>
<td>610</td>
</tr>
<tr>
<td>Boeing 747</td>
<td>470</td>
<td>4150</td>
<td>610</td>
</tr>
<tr>
<td>BAC/Sud Concorde</td>
<td>132</td>
<td>4000</td>
<td>1350</td>
</tr>
<tr>
<td>Douglas DC-8-50</td>
<td>146</td>
<td>8720</td>
<td>544</td>
</tr>
</tbody>
</table>

Which airplane has the best performance?
- How faster is the Concorde compared to B747?
- How much bigger is B747 than the DC-8?

<table>
<thead>
<tr>
<th>Plane</th>
<th>DC to Paris</th>
<th>Speed</th>
<th>Passengers</th>
<th>Throughput (p.mph)</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>Concorde</td>
<td>3 hours</td>
<td>1350 mph</td>
<td>132</td>
<td>178,200</td>
</tr>
</tbody>
</table>
Computer Performance

Computer Performance is related to TIME, TIME and TIME

Two notions of Performance

- Time to do the task:
  - Execution time, Response time (latency)
    - How long does it take for a job to run?
    - How long does it take to execute a job?
    - How long must I wait for the database query?

- Tasks per day, hour, week, sec, nsec, etc.
  - How many jobs can a machine run at once?
  - What is the average execution rate?

- When we upgrade a Pentium-IV PC with a new i7 quad core processor:
  What do we increase?

- When we add a new computer system to the lab:
  What do we increase?
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…
CPU Clocking

Operation of digital hardware governed by a constant-rate of clock

Clock period: duration of a clock cycle
e.g. 250ps = 0.25ns = 250×10^{-10}ns

Clock frequency (rate): cycles per second
e.g. 4.0GHz = 4000MHz = 4.0×10^9Hz
Execution Time

The execution time is defined in terms of:

Elapsed Time
Counts everything

A useful number, but often not good for comparison purposes.

CPU time
Doesn't count I/O or time spent running other programs.

The user CPU time
The time spent executing the lines of code that are "in" our program.

Clock Cycles
Instead of reporting execution time in seconds,

We often use cycles

\[
\text{seconds per program} = \frac{\text{cycles per program}}{\text{cycles per cycle}} \times \text{seconds per cycle}
\]

An 800 MHz. clock has a cycle time of

\[
\frac{1}{800 \times 10^6} \times 10^9 = 1.25 \text{ nanosec}
\]
Basic Definition of Performance

For some program running on machine X,
\[(\text{Performance})_X = 1 / (\text{Execution time})_X\]
When X is \(n\) times faster than Y machine
\[(\text{Performance})_X / (\text{Performance})_Y = n\]

**Problem:**
Machine A runs a program in 20 seconds
Machine B runs the same program in 25 seconds

**How to Improve Performance**
Everything else being equal we can either:
- Reduce the number of required cycles for a program, or
- Reduce the clock cycle time or, said another way, the clock rate.

Hardware designer often trade off clock rate against cycle count

**Can we assume: # of cycles = # of instructions?**
- Multiplication takes more time than addition.
- Floating-point operations take longer than integer.
- Accessing memory takes more time than registers.
CPU Time
Proportional to Instruction Count

\[(\text{CPU-time/Program}) = ?? \ (\text{Instructions/Program})\]

When ISA is set, what can influence instruction count?

**Machine Instructions:**
Static count?
or dynamic count?

**Program:**
What type of computer architect influences the number of instructions, a given program needs?

Any additional instruction you execute takes time.

**CPU time: Proportional to Clock Period**
How can architects reduce clock period?

**Instruction’s exe time in “number of cycles”**.
Short clock period => Short execution time.

What ultimately limits an architect’s ability to reduce the clock period?
CPU Time Example

Computer $A$: 2GHz clock, 10-sec CPU time

Designing Computer $B$
- Aim for a 6-sec CPU time
- Can have faster clock, but causes $1.2 \times$ clock cycles

How fast must Computer $B$ clock be?

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

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

$= 10s \times 2GHz = 20 \times 10^9$

Clock Rate$_B = \frac{1.2 \times 20 \times 10^9}{6s} = \frac{24 \times 10^9}{6s} = 4GHz$
# Aspects of CPU Performance

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Instruction_count</th>
<th>CPI</th>
<th>Clock_cycle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Programming Language</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Compiler</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>ISA</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Core organization</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Technology</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**CPI:** Cycles per Instruction (average)

\[
\text{CPI} = \frac{(\text{CPU Time} \times \text{Clock Rate})}{\text{Instruction Count}}
\]

\[
\text{CPU time} = \text{ClockCycleTime} \times \sum_{j=1}^{n} \text{CPI}_j \times I_j
\]

\[
\text{CPI} = \sum_{j=1}^{n} \text{CPI}_j \times F_j \quad \text{where } F \text{ is instruction frequency}
\]

\[
\text{and } F_j = \frac{I_j}{\text{(instruction count)}}
\]
Performance Equation

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

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

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

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: Analytical Tool to Design

Program Instruction

Machine CPI

\[
5 \times 30 + 1 \times 20 + 2 \times 20 + 2 \times 10 + 2 \times 20 = 100
\]
CPI Example

Suppose we have two implementations of the same instruction set architecture (ISA). For some program,

Machine A has a clock cycle time of 250 psec and average CPI of 2.0
Machine B has a clock cycle time of 400 psec and average CPI of 1.2

Which machine is faster for this program, and by how much?

If two machines have the same ISA which of the quantities (e.g. clock rate, CPI, execution time, # of instructions, MIPS) will always be identical?
Number of Instructions: Example

A compiler designer is trying to decide between two code sequences for a particular machine. Based on the hardware implementation, there are three different classes of instructions: Class A, Class B, and Class C, and they require one, two, and three cycles (respectively).

The first code sequence has 5 instructions: 2 of A, 1 of B, and 2 of C
The second sequence has 6 instructions: 4 of A, 1 of B, and 1 of C.

Which sequence will be faster? How much? What is the CPI for each sequence?
MIPS and MFLOPS

MIPS is often used as an alternative to time for indicating performance.

\[
\text{MIPS} = \frac{\text{Instruction count}}{(\text{Execution time} \times 10^6)}
\]

This is also called native MIPS. Faster machine will have higher MIPS.

Mainly three problems with MIPS

- It does not take into account the capabilities of instructions. You cannot compare two computers with different instruction sets.
- MIPS will vary for different programs on the same machine.
- MIPS can vary inversely with performance.

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

➤ CPI varies between programs on a given CPI.
An Example

Two different compilers are tested for a 1 GHz computer with three classes of instructions:

- Class A instructions require one cycle
- Class B instructions have two cycle
- Class C require three cycles

Both compilers are used to produce a code for large piece of software. First compiler's code uses:

- 5 million Class A instructions
- 1 million Class B instructions
- 1 million Class C instructions.

The second compiler's code uses:

- 10 million Class A instructions
- 1 million Class B instructions
- 1 million Class C instructions.

Which sequence will be faster?
Benchmarks

Performance best determined by running a real application
- Use programs typical of expected workload.
- Or, typical of expected class of applications.

Small Benchmarks
- Nice for architects and designers
- Easy to standardize
- Can be abused

SPEC
(System Performance Evaluation Corporation)
- System/CPU Manufacturers and others have agreed on a set of real program and inputs
- Can still be abused (Intel’s “other” bug)

Intel compiler generated wrong code for Pentium showing huge performance gain.

- Valuable indicator of performance.
Benchmark Games

Saturday, January 6, 1996 New York Times

An embarrassed Intel Corp. acknowledged Friday that a bug in a software program known as a compiler had led the company to overstate the speed of its microprocessor chips on an industry benchmark by 10 percent. However, industry analysts said the coding error...was a sad commentary on a common industry practice of “cheating” on standardized performance tests...The error was pointed out to Intel two days ago by a competitor, Motorola ...came in a test known as SPECint92...Intel acknowledged that it had “optimized” its compiler to improve its test scores. The company had also said that it did not like the practice but felt to compelled to make the optimizations because its competitors were doing the same thing...At the heart of Intel’s problem is the practice of “tuning” compiler programs to recognize certain computing problems in the test and then substituting special handwritten pieces of code...
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 ’95: Based on real programs

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>go</td>
<td>Artificial intelligence; plays the game of Go</td>
</tr>
<tr>
<td>m88ksim</td>
<td>Motorola 88k chip simulator; runs test program</td>
</tr>
<tr>
<td>gcc</td>
<td>The Gnu C compiler generating SPARC code</td>
</tr>
<tr>
<td>compress</td>
<td>Compresses and decompresses file in memory</td>
</tr>
<tr>
<td>li</td>
<td>Lisp interpreter</td>
</tr>
<tr>
<td>iijpeg</td>
<td>Graphic compression and decompression</td>
</tr>
<tr>
<td>perl</td>
<td>Manipulates strings and prime numbers in the special-purpose programming language Perl</td>
</tr>
<tr>
<td>vortex</td>
<td>A database program</td>
</tr>
<tr>
<td>tomcatv</td>
<td>A mesh generation program</td>
</tr>
<tr>
<td>swim</td>
<td>Shallow water model with 513 x 513 grid</td>
</tr>
<tr>
<td>su2cor</td>
<td>Quantum physics; Monte Carlo simulation</td>
</tr>
<tr>
<td>hydro2d</td>
<td>Astrophysics; Hydrodynamic Naiver Stokes equations</td>
</tr>
<tr>
<td>mgrid</td>
<td>Multigrid solver in 3-D potential field</td>
</tr>
<tr>
<td>applu</td>
<td>Parabolic/elliptic partial differential equations</td>
</tr>
<tr>
<td>trub3d</td>
<td>Simulates isotropic, homogeneous turbulence in a cube</td>
</tr>
<tr>
<td>apsi</td>
<td>Solves problems regarding temperature, wind velocity, and distribution of pollutant</td>
</tr>
<tr>
<td>fpppp</td>
<td>Quantum chemistry</td>
</tr>
<tr>
<td>wave5</td>
<td>Plasma physics; electromagnetic particle simulation</td>
</tr>
</tbody>
</table>

Main Sources for CPU performance improvement.
  • Clock rate
  • CPI due to processor organization
  • Compiler enhancement
### SPEC CPU2000

<table>
<thead>
<tr>
<th>Integer benchmarks</th>
<th>FP benchmarks</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>gzip</td>
<td>Compression</td>
</tr>
<tr>
<td>vpr</td>
<td>FPGA circuit placement and routing</td>
</tr>
<tr>
<td>gcc</td>
<td>The Gnu C compiler</td>
</tr>
<tr>
<td>mcf</td>
<td>Combinatorial optimization</td>
</tr>
<tr>
<td>crafty</td>
<td>Chess program</td>
</tr>
<tr>
<td>parser</td>
<td>Word processing program</td>
</tr>
<tr>
<td>eon</td>
<td>Computer visualization</td>
</tr>
<tr>
<td>perlbmk</td>
<td>perl application</td>
</tr>
<tr>
<td>gap</td>
<td>Group theory, Interpreter</td>
</tr>
<tr>
<td>vortex</td>
<td>Object-oriented database</td>
</tr>
<tr>
<td>bzip2</td>
<td>Compression</td>
</tr>
<tr>
<td>twolf</td>
<td>Place and route simulator</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**FIGURE 4.5 The SPEC CPU2000 benchmarks.** The 12 integer benchmarks in the left half of the table are written in C and C++, while the floating-point benchmarks in the right half are written in Fortran (77 or 90) and C. For more information on SPEC and on the SPEC benchmarks, see www.spec.org. The SPEC CPU benchmarks use wall clock time as the metric, but because there is little I/O, they measure CPU performance.
**SPEC CPU2000**

*Does doubling the clock rate double the performance? Can a machine with a slower clock rate have better performance?*

![Graph showing SPEC CPU2000 performance vs clock rate.](image-url)

- **Pentium 4 CFP2000**
- **Pentium 4 CINT2000**
- **Pentium III CFP2000**
- **Pentium III CINT2000**

**Legend:**
- **Pentium M @ 1.6/0.6 GHz**
- **Pentium 4-M @ 2.4/1.2 GHz**
- **Pentium III-M @ 1.2/0.8 GHz**
### SPEC CPU2006

Elapsed time to execute a set of programs

- Negligible I/O, so focuses on CPU performance

Normalize relative to reference machine.

Summarize as geometric mean of performance ratios:

CINT2006 (integer), CFP2006 (floating-point)

\[
\sqrt[n]{\prod_{i=1}^{n} \text{Execution time ratio}_i}
\]

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>perf</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
Processor Evaluation Basis

Pros

- representative
- portable
- widely used
- easy to run, early in design cycle
- identify peak capability and potential

Cons

- very specific
- non-portable
- difficult to run, or measure
- hard to identify
- “peak” may be a long way from application performance

Actual Target Workload

Full Application

Small Kernel Benchmark

Micro Benchmarks
Performance Metrics

Each metric has a place and a purpose, and each can be misused

- (millions) of Instructions per second
  * MIPS
- (millions) of (F.P.) operations per second
  * MFLOP/s
- Megabytes per second
- Cycles per second
  (clock rate)
- Useful Operations per second
  Answers per month

ISA
Datapath
Control
Function Units
Transistors Wires Pins

Compiler
Programming Language
Application
Amdahl's Law

Speedup due to enhancement E:

\[
\text{Speedup}(E) = \frac{\text{ExeTime w/o E}}{\text{ExeTime w/ E}} = \frac{\text{Performance w/ E}}{\text{Performance w/o E}}
\]

Suppose that enhancement E accelerates a fraction F of the task by a factor S and the remainder of the task is unaffected then,

\[
\text{ExTime(with E)} = ((1-F) + F/S) * \text{ExTime(without E)}
\]

\[
\text{Speedup(with E)} = \frac{1}{((1-F) + F/S)}
\]

Example: "Suppose a program runs in 100 seconds on a machine, with multiply responsible for 80 seconds of this time. How much do we have to improve the speed of multiplication if we want the program to run 4 times faster?"
Amdahl's Law

Example

Suppose we enhance a machine making all floating-point instructions run five times faster. If the execution time of some benchmark before the floating-point enhancement is 10 seconds, what will the speedup be if half of the 10 seconds is spent executing floating-point instructions?

We are looking for a benchmark to show off the new floating-point unit described above, and want the overall benchmark to show a speedup of 3. One benchmark we are considering runs for 100 seconds with the old floating-point hardware. How much of the execution time would floating-point instructions have to account for in this program in order to yield our desired speedup on this benchmark?
Amdahl’s Law
(of Diminishing Returns)

Where a program spends its time during execution

If enhancement “E” speeds up multiply, but other instructions are unchanged, what is the maximum speedup $S$?

\[
\text{Speedup(with E)} = \frac{1}{((1-F) + F/S)}
\]

\[
\text{Speedup(with E)} = \frac{1}{((1-0.5) + 0.5/\text{Max})}
\]

What is the lesson of Amdahl’s Law?
Enhancement by Multiple CPUs

Program We Wish to Run on $n$ CPUs

The program spends 30% of its time running code that can not be recoded to run in parallel.

Compute speedup for $N = 2, 3, 4, 5, \text{ and } \infty$

\[
\text{Speedup (with E)} \hspace{1em} \hat{\hat{S}} = \frac{1}{((1-F) + F/S)}
\]

\[
\text{Speedup (with E)} \hspace{1em} \hat{\hat{S}} = \frac{1}{((1-0.7) + 0.7/2)}
\]

\[
\text{Speedup (with E)} \hspace{1em} \hat{\hat{S}} = 1.54
\]

<table>
<thead>
<tr>
<th>CPUs</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>$\infty$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speedup</td>
<td>1.54</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Experimental Example

Phone a major computer retailer like Dell or MDG and tell them you are having trouble deciding between two different computers, specifically you are confused about the processors strengths and weaknesses

e.g.,
(Pentium 4 at 2Ghz vs. Celeron M at 1.4 Ghz )

• What kind of responses are you likely to get?
• What kind of response could you give a friend with the same question?
Points to Remember

Performance is specific to particular program(s)
- Execution time is a consistent summary of performance.

For a given architecture, performance increases due to:
- Increases in clock rate (without adverse CPI)
- Improvements in processor organization for lowering CPI.
- Compiler enhancements that lower CPI and/or instruction count.

\[
\text{CPU time} = \frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}
\]

Machines are Optimized with respect to program loads.

CPI of the program. Reflects the program’s instruction mix.

Clock period. Optimize jointly with machine CPI