**Performance of Computer Systems**

Dr. Arjan Durresi  
Louisiana State University  
Baton Rouge, LA 70810  
Durresi@Csc.LSU.Edu

These slides are available at:  
http://www.csc.lsu.edu/~durresi/CSC3501_07/

---

**Overview**

- Metrics
- How to improve performance?
- CPI
- MIPS
- Benchmarks

---

**The Design Process**

*“To Design Is To Represent”*

Design activity yields description/representation of an object

- Traditional craftsman does not distinguish between the conceptualization and the artifact
- Separation comes about because of complexity
- The concept is captured in one or more representation languages
- This process IS design

Design Begins With Requirements

- Functional Capabilities: what it will do
- Performance Characteristics: Speed, Power, Area, Cost, ...

---

**Design Process (cont.)**

Design Finishes As Assembly

- Design understood in terms of components and how they have been assembled
- Top Down decomposition of complex functions (behaviors) into more primitive functions
- Bottom-up composition of primitive building blocks into more complex assemblies

Design is a “creative process,” not a simple method

---

**Design Refinement**

Informal System Requirement

Initial Specification

Intermediate Specification

- Final Architectural Description

Intermediate Specification of Implementation

- Final Internal Specification

Physical Implementation

---

**Design as Search**

- Problem A
- Strategy 1
- SubProb1
- BB1

- Strategy 2
- SubProb2
- BB2

- SubProb3
- BB3

Design involves educated guesses and verification

- Given the goals, how should these be prioritized?
- Given alternative design pieces, which should be selected?
- Given design space of components & assemblies, which part will yield the best solution?

Feasible (good) choices vs. Optimal choices
Measurement and Evaluation

Architecture is an iterative process
-- searching the space of possible designs
-- at all levels of computer systems

Creative Ideas
Good Ideas
Bad Ideas
Mediocre Ideas

Performance

- Measure, Report, and Summarize
- Make intelligent choices
- See through the marketing hype
- Key to understanding underlying organizational motivation

Why is some hardware better than others for different programs?

What factors of system performance are hardware related?
(e.g., Do we need a new machine, or a new operating system?)

How does the machine’s instruction set affect performance?

Performance Metrics

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

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

- Both require
  - basis for comparison
  - metric for evaluation

Our goal is to understand what factors in the architecture contribute to overall system performance and the relative importance (and cost) of these factors.

Performance: Basic Metrics

- Response Time (latency)
  - How long does it take for my job to run?
  - How long does it take to execute a job?
  - How long must I wait for the database query?

- Throughput
  - How many jobs can the machine run at once?
  - What is the average execution rate?
  - How much work is getting done?

- Example: Car assembly factory:
  - 4 hours to produce a car (response time)
  - 6 cars per an hour produced (throughput)

- If we upgrade a machine with a new processor what do we increase?

- If we add a new machine to the lab what do we increase?

Computer Performance: Introduction

- The computer user is interested in response time (or execution time) - the time between the start and completion of a given task (program).
- The manager of a data processing center is interested in throughput - the total amount of work done in a given time.
- The computer user wants response time to decrease, while the manager wants throughput increased.
- Main factors influencing performance of computer system are:
  - processor and memory,
  - input/output controllers and peripherals,
  - compilers, and
  - operating system.

Computer Performance: Introduction

- The computer user is interested in response time (or execution time) - the time between the start and completion of a given task (program).
- The manager of a data processing center is interested in throughput - the total amount of work done in a given time.
- The computer user wants response time to decrease, while the manager wants throughput increased.
- Main factors influencing performance of computer system are:
  - processor and memory,
  - input/output controllers and peripherals,
  - compilers, and
  - operating system.

Performance: Basic Metrics

- Response Time (latency)
  - How long does it take for my job to run?
  - How long does it take to execute a job?
  - How long must I wait for the database query?

- Throughput
  - How many jobs can the machine run at once?
  - What is the average execution rate?
  - How much work is getting done?

- Example: Car assembly factory:
  - 4 hours to produce a car (response time)
  - 6 cars per an hour produced (throughput)

- If we upgrade a machine with a new processor what do we increase?

- If we add a new machine to the lab what do we increase?

Performance: Basic Metrics

- Response Time (latency)
  - How long does it take for my job to run?
  - How long does it take to execute a job?
  - How long must I wait for the database query?

- Throughput
  - How many jobs can the machine run at once?
  - What is the average execution rate?
  - How much work is getting done?

- Example: Car assembly factory:
  - 4 hours to produce a car (response time)
  - 6 cars per an hour produced (throughput)

- If we upgrade a machine with a new processor what do we increase?

- If we add a new machine to the lab what do we increase?

Performance: Basic Metrics

- Response Time (latency)
  - How long does it take for my job to run?
  - How long does it take to execute a job?
  - How long must I wait for the database query?

- Throughput
  - How many jobs can the machine run at once?
  - What is the average execution rate?
  - How much work is getting done?

- Example: Car assembly factory:
  - 4 hours to produce a car (response time)
  - 6 cars per an hour produced (throughput)

- If we upgrade a machine with a new processor what do we increase?

- If we add a new machine to the lab what do we increase?
**Execution Time**

- Elapsed Time
  - counts everything (disk and memory accesses, I/O, etc.)
  - a useful number, but often not good for comparison purposes
- CPU time
  - doesn't count I/O or time spent running other programs
  - can be broken up into system time, and user time
- Our focus: user CPU time
  - time spent executing the lines of code that are "in" our program
  - CPU time is a true measure of processor/memory performance.
  - Performance of processor/memory = 1 / CPU_time

**Book's Definition of Performance**

- For some program running on machine X,
  - PerformanceX = 1 / Execution timeX
- "X is n times faster than Y"
  - PerformanceX / PerformanceY = n
- Problem:
  - machine A runs a program in 20 seconds
  - machine B runs the same program in 25 seconds

**Analysis of CPU Time**

- CPU time depends on the program which is executed, including:
  - a number of instructions executed,
  - types of instructions executed and their frequency of usage.
- Computers are constructed in such a way that events in hardware are synchronized using a clock.
- Clock rate is given in Hz (=1/sec).
- A clock rate defines durations of discrete time intervals called clock cycle times or clock cycle periods:
  - clock_cycle_time = 1/clock_rate (in sec)
- Thus, when we refer to different instruction types (from performance point of view), we are referring to instructions with different number of clock cycles required (needed) to execute.

**Clock Cycles**

- Instead of reporting execution time in seconds, we often use cycles:
  - CPU time = seconds / cycles
  - cycle time = time between ticks = seconds per cycle
  - clock rate (frequency) = cycles per second (1 Hz = 1 cycle/sec)
  - A 4 GHz clock has a cycle time of
    \[ \frac{1}{4 \times 10^9} = \frac{1}{250} \text{ picoseconds (ps)} \]

**Clock Cycles (cont.)**

- Clock rate (MHz, GHz) is inverse of clock cycle time (clock period)
  \[ CC = \frac{1}{CR} \]

<table>
<thead>
<tr>
<th>Cycle Period</th>
<th>Clock Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 nsec</td>
<td>100 MHz</td>
</tr>
<tr>
<td>5 nsec</td>
<td>200 MHz</td>
</tr>
<tr>
<td>2 nsec</td>
<td>500 MHz</td>
</tr>
<tr>
<td>1 nsec</td>
<td>1 GHz</td>
</tr>
<tr>
<td>500 psec</td>
<td>2 GHz</td>
</tr>
<tr>
<td>250 psec</td>
<td>4 GHz</td>
</tr>
<tr>
<td>200 psec</td>
<td>5 GHz</td>
</tr>
</tbody>
</table>

**How to Improve Performance**

- So, to improve performance (everything else being equal) you can either (increase or decrease?)
  - ______ the # of required cycles for a program, or
  - ______ the clock cycle time or, said another way, ______ the clock rate.
Example

- Our favorite program runs in 10 seconds on computer A, which has a 4 GHz clock. We are trying to help a computer designer build a new machine B, that will run this program in 6 seconds. The designer can use new (or perhaps more expensive) technology to substantially increase the clock rate, but has informed us that this increase will affect the rest of the CPU design, causing machine B to require 1.2 times as many clock cycles as machine A for the same program. What clock rate should we tell the designer to target?

- Don’t Panic, can easily work this out from basic principles.

Example

- A program runs in 10s on computer A, at 4GHz
- How to build a computer B to run this program in 6s
- The designer has determined that if the clock rate will be increased, it will cause computer B to require 1.2 times more clock cycles than A
- What clock rate should be used in computer B?

Don’t Panic, can easily work this out from basic principles.

### Measuring Time using Clock Cycles

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

- One way to define clock cycles:
  
  \[
  \text{Clock cycles for program} = (\text{Instructions for a program (called "Instruction Count")} \times \text{Average Clock cycles Per Instruction (called "CPI")})
  \]

- CPI - the average number of clock cycles per instruction is an important parameter

- \[\text{CPI} = \frac{\text{Clock cycles for a program}}{\text{Instruction count}}\]

- **Instruction count** is the number of instructions executed.

### Performance Calculation

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

- Substituting for clock cycles
  
  \[
  \text{CPU time} = \frac{\text{CPU clock cycles}}{\text{Clock rate}}
  \]

- **CPU time** = \(\frac{\text{Program instruction count} \times \text{CPI}}{\text{Clock cycle time}}\)

### How Calculate the 3 Components?

- **Clock Cycle Time:** in specification of computer (Clock Rate in advertisements)
- **Instruction Count:**
  
  - Count instructions in loop of small program
  - Use simulator to count instructions
  - Hardware counter in spec. register (most CPUs)
- **CPI:** Clock cycles for a program/Instruction count
  
  - Calculate Execution Time / Clock cycle time
  - Hardware counter in special register (most CPUs)

### How many cycles are required for a program?

- Could assume that number of cycles equals number of instructions

This assumption is incorrect, different instructions take different amounts of time on different machines.

Why? hint: remember that these are machine instructions, not lines of C code
Different numbers of cycles for different instructions

- Multiplication takes more time than addition
- Floating point operations take longer than integer ones
- Accessing memory takes more time than accessing registers
- Important point: changing the cycle time often changes the number of cycles required for various instructions (more later)

Phases in Instruction Execution

- We can divide the execution of an instruction into the following five stages:
  - Instruction fetch
  - Instruction decode and register fetch
  - Execution, effective address or branch calculation
  - Memory access (for lw and sw instructions only)
  - Register write back (for ALU and lw instructions)

Sequential Execution of 3 LW Instructions

- Assumed are the following delays: Memory access = 2 nsec, ALU operation = 2 nsec, Register file access = 1 nsec;

<table>
<thead>
<tr>
<th>Program Execution order</th>
<th>m</th>
<th>s</th>
<th>m</th>
</tr>
</thead>
<tbody>
<tr>
<td>lw r1, 100[r0]</td>
<td></td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>lw r2, 200[r0]</td>
<td></td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>lw r2, 200[r0]</td>
<td></td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

Every lw instruction needs 8 nsec to execute.

In this course, we are designing processor that executes instructions sequentially.

Performance

- Performance is determined by execution time
- Do any of the other variables equal performance?
  - # of cycles to execute program?
  - # of instructions in program?
  - # of cycles per second?
  - average # of cycles per instruction?
  - average # of instructions per second?
- Common pitfall: thinking one of the variables is indicative of performance when it really isn’t.

Now that we understand cycles

- A given program will require:
  - some number of instructions (machine instructions)
  - some number of cycles
  - some number of seconds
- We have a vocabulary that relates these quantities:
  - cycle time (seconds per cycle)
  - clock rate (cycles per second)
  - CPI (cycles per instruction)
    a floating point intensive application might have a higher CPI
  - MIPS (millions of instructions per second)
    this would be higher for a program using simple instructions

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 ps and a CPI of 2.0
    Machine B has a clock cycle time of 500 ps and a CPI of 1.2
  - What machine is faster for this program, and by how much?

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

CPU clock cycles\textsubscript{a} = I \times 2.0 \cdot CPU clock cycles\textsubscript{b} = I \times 1.2

CPU time\textsubscript{a} = CPU clock cycles\textsubscript{a} \times CPU clock time\textsubscript{a} = I \times 2.0 \times 250ps = I \times 500ps
CPU time\textsubscript{b} = CPU clock cycles\textsubscript{b} \times CPU clock time\textsubscript{b} = I \times 1.2 \times 500ps = I \times 600ps

CPU time\textsubscript{a} = 1.2

CPU time = \frac{\text{Instruction count} \times CPI}{\text{Clock rate}}

CPI

- CPU clock cycles = \sum (\text{CPI}_i \times C_i)
- C_i is the count of the number of instructions of class i, CPI is the average number per instructions for that class.

Computer Performance

\begin{tabular}{|c|c|c|c|}
\hline
\textbf{Program} & \textbf{Inst. Count} & \textbf{CPI} & \textbf{Clock Rate} \\
\hline
Program & X & & \\
Compiler & X & (X) & \\
Inst. Set. & X & X & \\
Organization & X & X & \\
Technology & X & X & \\
\hline
\end{tabular}

# of Instructions Example

- CPU clock cycles\textsubscript{1} = \sum (\text{CPI}_1 \times C_i) = (2x1)+(1x2)+(2x3) = 10 cycles
- CPU clock cycles\textsubscript{2} = \sum (\text{CPI}_2 \times C_i) = (4x1)+(1x2)+(1x3) = 9 cycles
- CPI\textsubscript{1} = 10/2 = 2
- CPI\textsubscript{2} = 9/6 = 1.5
- When comparing, all three factors: clock rate, number of instructions, and CPI should be compared.

CPU Time: Example

- Consider an implementation of MIPS ISA with 500 MHz clock
- - each ALU instruction takes 3 clock cycles
- - each branch/jump instruction takes 2 clock cycles
- - each sw instruction takes 4 clock cycles
- - each lw instruction takes 5 clock cycles
- Also, consider a program that during its execution executes:
  - x = 200 million ALU instructions
  - y = 55 million branch/jump instructions
  - z = 25 million sw instructions
  - w = 20 million lw instructions
- Find CPU time.
CPU Time: Example 1 (continued)

- Approach 1:
  \[ \text{Clock cycles for a program} = (x \times 3 + y \times 2 + z \times 4 + w \times 5) = 910 \times 10^{6} \text{ clock cycles} \]
  \[ \text{CPU time} = \frac{\text{Clock cycles for a program}}{\text{Clock rate}} = \frac{910 \times 10^{6}}{500 \times 10^{6}} = 1.82 \text{ sec} \]

- Approach 2:
  \[ \text{CPI} = \frac{\text{Clock cycles for a program}}{\text{Instructions count}} \]
  \[ \text{CPU time} = \text{Instruction count} \times \text{CPI} / \text{Clock rate} = \frac{(x+y+z+w) \times 3.03}{500 \times 10^{6}} = 1.82 \text{ sec} \]

Analysis of CPU Performance Equation

- CPU time = Instruction count \times CPI / Clock rate
- How to improve (i.e. decrease) CPU time:
  - Clock rate: hardware technology & organization,
  - CPI: organization, ISA and compiler technology,
  - Instruction count: ISA & compiler technology.
- Many potential performance improvement techniques primarily improve one component with small or predictable impact on the other two.

Calculating Components of CPU time

- For an existing processor it is easy to obtain the CPU time (i.e. the execution time) by measurement, and the clock rate is known. But, it is difficult to figure out the instruction count or CPI.
- Newer processors, MIPS64 processor is such an example include counters for instructions executed and for clock cycles. Those can be helpful to programmers trying to understand and tune the performance of an application.
- Also, different simulation techniques and queuing theory could be used to obtain values for components of the execution (CPU) time.

Attempting to Calculate CPI

- The table below indicates frequency of all instruction types executed in a "typical" program and, from the reference manual, we are provided with a number of cycles per instruction for each type.

<table>
<thead>
<tr>
<th>Instruction Type</th>
<th>Frequency</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU instruction</td>
<td>50%</td>
<td>4</td>
</tr>
<tr>
<td>Load instruction</td>
<td>30%</td>
<td>5</td>
</tr>
<tr>
<td>Store instruction</td>
<td>5%</td>
<td>4</td>
</tr>
<tr>
<td>Branch instruction</td>
<td>15%</td>
<td>2</td>
</tr>
</tbody>
</table>

\[ \text{CPI} = 0.5 \times 4 + 0.3 \times 5 + 0.05 \times 4 + 0.15 \times 2 = 4 \text{ cycles/instruction} \]

- The calculation may not be necessary correct since the numbers for cycles per instruction given don’t account for pipeline effects.

A Simple Example

<table>
<thead>
<tr>
<th>Op</th>
<th>Freq</th>
<th>CPI</th>
<th>Freq x CPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU</td>
<td>50%</td>
<td>1</td>
<td>50</td>
</tr>
<tr>
<td>Load</td>
<td>20%</td>
<td>5</td>
<td>100</td>
</tr>
<tr>
<td>Store</td>
<td>10%</td>
<td>3</td>
<td>30</td>
</tr>
<tr>
<td>Branch</td>
<td>20%</td>
<td>2</td>
<td>40</td>
</tr>
</tbody>
</table>

\[ \Sigma = \]
A Simple Example

<table>
<thead>
<tr>
<th>Op</th>
<th>Freq</th>
<th>CPI</th>
<th>Freq x CPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU</td>
<td>50%</td>
<td>1</td>
<td>.5</td>
</tr>
<tr>
<td>Load</td>
<td>20%</td>
<td>5</td>
<td>.1</td>
</tr>
<tr>
<td>Store</td>
<td>10%</td>
<td>3</td>
<td>.3</td>
</tr>
<tr>
<td>Branch</td>
<td>20%</td>
<td>2</td>
<td>.4</td>
</tr>
</tbody>
</table>

Σ = 2.2

How much faster would the machine be if a better data cache reduced the average load time to 2 cycles?
- CPU time new = 1.6 x IC x CC
- 2.2/1.6 means 37.5% faster

How does this compare with using branch prediction to shave a cycle off the branch time?
- CPU time new = 2.0 x IC x CC
- 2.2/2.0 means 10% faster

What if two ALU instructions could be executed at once?
- CPU time new = 1.95 x IC x CC
- 2.2/1.95 means 12.8% faster

Pipelining: Its Natural!

- Laundry Example.
- Ann, Brian, Cathy, Dave each have one load of clothes to wash, dry, and fold
- Washer takes 30 minutes
- Dryer takes 40 minutes
- "Folder" takes 20 minutes

Sequential Laundry

<table>
<thead>
<tr>
<th>Time</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 PM</td>
<td>30</td>
<td>40</td>
<td>40</td>
<td>40</td>
</tr>
<tr>
<td>7 PM</td>
<td>40</td>
<td>40</td>
<td>40</td>
<td>40</td>
</tr>
<tr>
<td>8 PM</td>
<td>40</td>
<td>40</td>
<td>40</td>
<td>40</td>
</tr>
<tr>
<td>9 PM</td>
<td>40</td>
<td>40</td>
<td>40</td>
<td>40</td>
</tr>
<tr>
<td>10 PM</td>
<td>40</td>
<td>40</td>
<td>40</td>
<td>40</td>
</tr>
<tr>
<td>11 PM</td>
<td>40</td>
<td>40</td>
<td>40</td>
<td>40</td>
</tr>
</tbody>
</table>

- Sequential laundry takes 6 hours for 4 loads
- If they learned pipelining, how long would laundry take?

Pipelined Laundry

<table>
<thead>
<tr>
<th>Time</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 PM</td>
<td>30</td>
<td>40</td>
<td>40</td>
<td>40</td>
</tr>
<tr>
<td>7 PM</td>
<td>40</td>
<td>40</td>
<td>40</td>
<td>40</td>
</tr>
<tr>
<td>8 PM</td>
<td>40</td>
<td>40</td>
<td>40</td>
<td>40</td>
</tr>
<tr>
<td>9 PM</td>
<td>40</td>
<td>40</td>
<td>40</td>
<td>40</td>
</tr>
<tr>
<td>10 PM</td>
<td>40</td>
<td>40</td>
<td>40</td>
<td>40</td>
</tr>
<tr>
<td>11 PM</td>
<td>40</td>
<td>40</td>
<td>40</td>
<td>40</td>
</tr>
</tbody>
</table>

- Pipelined laundry takes 3.5 hours for 4 loads

Pipelining Lessons

- Pipelining doesn't help latency of single task, it helps throughput of entire workload
- Pipeline rate limited by slowest pipeline stage
- Multiple tasks operating simultaneously
- Potential speedup = Number pipe stages
- Unbalanced lengths of pipe stages reduces speedup
- Time to "fill" pipeline and time to "drain" it reduces speedup

Computer Pipelines

- Execute billions of instructions, so throughput is what matters
- What is desirable in instruction sets for pipelining?
  - Variable length instructions vs. all instructions same length?
  - Memory operands part of any operation vs. memory operands only in loads or stores?
  - Register operand many places in instruction format vs. registers located in same place?
Pipeline Executing 3 LW Instructions

Assuming delays as in the sequential case and pipelined processor with a clock cycle time of 2 nsec:

- lw r1, 100(r0)
- lw r2, 200(r0)

- lw r2, 200(r0)

- Note that registers are written during the first part of a cycle and read during the second part of the same cycle.
- Pipelining doesn't help to execute a single instruction, it may improve performance by increasing instruction throughput.

MIPS

- One alternative to time is the metric MIPS (Million Instructions per Second)
- MIPS = Instruction count
- MIPS = Execution time x 10^6
- MIPS does not take into account the capabilities of instructions
- MIPS varies among programs on the same computer
- MIPS can vary inversely with performance

MIPS example

- Two different compilers are being tested for a 4 GHz machine with three different classes of instructions: Class A, Class B, and Class C, which require one, two, and three cycles (respectively). Both compilers are used to produce code for a large piece of software.

  The first compiler's code uses 5 million Class A instructions, 1 million Class B instructions, and 1 million Class C instructions.

  The second compiler's code uses 10 million Class A instructions, 1 million Class B instructions, and 1 million Class C instructions.

- Which sequence will be faster according to MIPS?
- Which sequence will be faster according to execution time?

MIPS example

- MIPS = Instruction count
- MIPS = Execution time x 10^6
- MIPS = 2800
- MIPS = 3200

Quantitative Performance Measures

- Another popular, misleading and essentially useless measure was peak MIPS. That is a MIPS obtained using an instruction mix that minimizes the CPI, even if that instruction mix is totally impractical. Computer manufacturers still occasionally announce products using peak MIPS as a metric, often neglecting to include the work “peak”.

  Another popular alternative to execution time was million floating point operations per second - MFLOPS.

  MFLOPS = Number of floating point operations in a program
  MFLOPS = Execution time x 10^6

- Because it is based on operations in the program rather than on instructions, MFLOPS has a stronger claim than MIPS to being a fair comparison between different machines. MFLOPS are not applicable outside floating-point performance.

Benchmarks

- Performance best determined by running a real application
  - Use programs typical of expected workload
  - Or use typical of expected class of applications, e.g., compilers/editors, scientific applications, graphics, etc.

  Small benchmarks
  - nice for architects and designers
  - easy to standardize
  - can be abused

  SPEC (System Performance Evaluation Cooperative) was founded in late 1980s
  - companies have agreed on a set of real program and inputs
  - valuable indicator of performance (and compiler technology)
  - can still be abused
SPEC Benchmark Suites

- The SPEC benchmarks are real programs, modified for portability and to minimize the role of I/O in overall benchmark performance. Example: Optimizer GNU C compiler.
- First in 1989, SPEC89 was introduced with 4 integer programs and 6 floating point programs, providing a single "SPECmark".
- SPEC92 had 5 integer programs and 14 floating point programs, and provided SPECint92 and SPECfp92.
- SPEC95 provided SPECint_base95, SPECfp_base95.
- SPEC CPU2000 has 12 integer benchmarks and 14 floating point benchmarks, and provides CINT2000 and CFP2000.

Benchmark Games

- 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 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...

Saturday, January 6, 1996 New York Times

SPEC '89

- Compiler "enhancements" and performance

SPEC CPU2000

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

SPEC 2000

- Always on/maximum clock
- Laptop mode/adaptive clock
- Minimum power/minimum clock

SPEC 2000

- CINT2000/Clock (MHz) 0.47
- CFP2000/Clock (MHz) 0.34

CINT 2000 – CPI of Pentium 4 is 1.3 times that of Pentium 3 (0.47/0.36)

How come these numbers are reversed for CFP?

Pentium 4 provides a new set of instructions (Streaming SIMD)
So both CPI and instruction count are different

SPEC CPU2000

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>IP benchmarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>gcc</td>
<td>bzip2</td>
</tr>
<tr>
<td>gpr</td>
<td>F20-042</td>
</tr>
<tr>
<td>gem</td>
<td>F20-024</td>
</tr>
<tr>
<td>gpr</td>
<td>F20-042</td>
</tr>
<tr>
<td>gcc</td>
<td>CINT2000</td>
</tr>
<tr>
<td>gcc</td>
<td>CFP2000</td>
</tr>
<tr>
<td>gcc</td>
<td>CINT2000</td>
</tr>
<tr>
<td>gcc</td>
<td>CFP2000</td>
</tr>
<tr>
<td>gcc</td>
<td>CINT2000</td>
</tr>
<tr>
<td>gcc</td>
<td>CFP2000</td>
</tr>
</tbody>
</table>

SPEC CPU2000

<table>
<thead>
<tr>
<th>Ratio</th>
<th>Pentium IV</th>
<th>Pentium IV</th>
</tr>
</thead>
<tbody>
<tr>
<td>CINT2000/Clock (MHz)</td>
<td>0.47</td>
<td>0.36</td>
</tr>
<tr>
<td>CFP2000/Clock (MHz)</td>
<td>0.34</td>
<td>0.39</td>
</tr>
</tbody>
</table>
We are interested in two implementations of two similar but still different ISA, one with and one without special real number instructions.

Both machines have 1000MHz clock.

Machine With Floating Point Hardware - MFP implements real number operations directly with the following characteristics:
- real number multiply instruction requires 6 clock cycles
- real number add instruction requires 4 clock cycles
- real number divide instruction requires 20 clock cycles
- Any other instruction (including integer instructions) requires 2 clock cycles

Machine with No Floating Point Hardware - MNFP does not support real number instructions, but all its instructions are identical to non-real number instructions of MFP. Each MNFP instruction (including integer instructions) takes 2 clock cycles. Thus, MNFP is identical to MFP without real number instructions.

Any real number operation (in a program) has to be emulated by an appropriate so\textit{\textbf{ftware}} subroutine (i.e. compiler has to insert an appropriate sequence of integer instructions for each real number operation). The number of integer instructions needed to implement each real number operation is as follows:
- real number multiply needs 30 integer instructions
- real number add needs 20 integer instructions
- real number divide needs 50 integer instructions

Consider Program \textit{\textbf{P}} with the following mix of operations:
- real number multiply 10%
- real number add 15%
- real number divide 5%
- other instructions 70%

A. Find MIPS rating for both machines.

\[
\text{CPIMFP} = 0.1 \times 6 + 0.15 \times 4 + 0.05 \times 20 + 0.7 \times 2 = 3.6 \text{ clocks/instr}
\]

\[
\text{CPIMNFP} = 2 \text{ clocks/instr}
\]

According to MIPS rating, MNFP is better than MFP?

B. If Program \textit{\textbf{P}} on MFP needs 300,000,000 instructions, find the time to execute this program on each machine.

\[
\text{CPU\_timeMFP} = 300 \times 10^6 \times 3.6 / 1000 \times 10^6 = 1.08 \text{ sec}
\]

\[
\text{CPU\_timeMNFP} = 5 \times 10^6 \times 2 / 1000 \times 10^6 = 5.52 \text{ sec}
\]

C. Calculate MFLOPS for both computers.

\[
\text{MFLOPS} = \frac{\text{Number of floating point operations in a program}}{\text{Execution time} \times 10^8}
\]

\[
\text{MFLOPS\_MFP} = 90 \times 10^8 / 1.08 \times 10^8 = 83.3
\]

\[
\text{MFLOPS\_MNFP} = 90 \times 10^8 / 5.52 \times 10^8 = 16.3
\]

Phone a major computer retailer 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 response are you likely to get?

What kind of response could you give a friend with the same question?
Summarizing Performance

• The arithmetic mean of the execution times is given as:

$$\frac{1}{n} \sum_{i=1}^{n} \text{Time}_i$$

where \( \text{Time}_i \) is the execution time for the \( i \)th program of a total of \( n \) in the workload.

• The weighted arithmetic mean of execution times is given as:

$$\frac{\sum_{i=1}^{n} \text{Weight}_i \times \text{Time}_i}{\sum_{i=1}^{n} \text{Weight}_i}$$

where \( \text{Weight}_i \) is the frequency of the \( i \)th program in the workload.

• The geometric mean of execution times is given as:

$$\left( \prod_{i=1}^{n} \text{Time}_i \right)^{1/n}$$

where \( \prod_{i=1}^{n} x_i = x_1 \cdot x_2 \cdot \ldots \cdot x_n \)

- The geometric mean is independent of which data series we use for normalization because it has the property:

$$\prod_{i=1}^{n} \frac{x_i}{y_i} = \frac{\prod_{i=1}^{n} x_i}{\prod_{i=1}^{n} y_i}$$

- The advantage of the geometric mean is that it is independent of the running times of the individual programs, and it doesn't matter which computer is used for normalization.

- The drawback to using geometric means of execution times is that they violate our fundamental principle of performance measurement—they do not predict execution time.

- The ideal solution is to measure a real workload and weight the programs according to their frequency of execution.

Geometric mean.

<table>
<thead>
<tr>
<th>Program</th>
<th>Time as ( x )</th>
<th>Time as ( y )</th>
<th>Normalized to 2</th>
<th>Normalized to 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program 1</td>
<td>200</td>
<td>300</td>
<td>1.25</td>
<td>1.5</td>
</tr>
<tr>
<td>Program 2</td>
<td>300</td>
<td>400</td>
<td>1.50</td>
<td>2.0</td>
</tr>
<tr>
<td>Program 3</td>
<td>400</td>
<td>500</td>
<td>1.50</td>
<td>2.5</td>
</tr>
</tbody>
</table>

$$\prod_{i=1}^{n} \text{Execution time ratio}_i$$

where Execution time ratio \( i \) is the execution time, normalized to the reference computer, for the \( i \)th program of a total of \( n \) in the workload, and

$$\prod_{i=1}^{n} a_i$$ means the product \( a_1 \cdot a_2 \cdot \ldots \cdot a_n \)

Mean

- The geometric mean is independent of which data series we use for normalization because it has the property:

$$\text{Geometric mean}(x) = \left( \prod_{i=1}^{n} x_i \right)^{1/n}$$

- The advantage of the geometric mean is that it is independent of the running times of the individual programs, and it doesn't matter which computer is used for normalization.

- The drawback to using geometric means of execution times is that they violate our fundamental principle of performance measurement—they do not predict execution time.

- The ideal solution is to measure a real workload and weight the programs according to their frequency of execution.
Amdahl’s Law

Execution Time After Improvement = 

\[ \text{Execution Time Unaffected} + \left( \frac{\text{Execution Time Affected}}{\text{Amount of Improvement}} \right) \]

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?”

How about making it 5 times faster?


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?

Remember

- Performance is specific to a particular program/s
- Total execution time is a consistent summary of performance
- For a given architecture performance increases come from:
  - increases in clock rate (without adverse CPI affects)
  - improvements in processor organization that lower CPI
  - compiler enhancements that lower CPI and/or instruction count
- Algorithm/Language choices that affect instruction count
- Pitfall: expecting improvement in one aspect of a machine’s performance to affect the total performance

The Art of Performance Evaluation:

The Ratio Game

If you can’t convince them, confuse them.

Truman’s Law

The two systems are equally good.
The Ratio Game 2

Throughput in Transaction per Second

<table>
<thead>
<tr>
<th>System</th>
<th>Workload 1</th>
<th>Workload 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>B</td>
<td>10</td>
<td>20</td>
</tr>
</tbody>
</table>

Throughput will Respect to System A

<table>
<thead>
<tr>
<th>System</th>
<th>Workload 1</th>
<th>Workload 2</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>0.5</td>
<td>2</td>
<td>1.25</td>
</tr>
</tbody>
</table>

System B is better than system A!!

The problem is with taking the average of ratios

Ratio Game with Percentages

<table>
<thead>
<tr>
<th>System</th>
<th>Test 1 Total</th>
<th>Pass</th>
<th>% Pass</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>300</td>
<td>60</td>
<td>20</td>
</tr>
<tr>
<td>B</td>
<td>350</td>
<td>62</td>
<td>20.6</td>
</tr>
</tbody>
</table>

Throughput will Respect to System A

<table>
<thead>
<tr>
<th>System</th>
<th>Test 1 Total</th>
<th>Pass</th>
<th>% Pass</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>8</td>
<td>25</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>40</td>
<td>8</td>
</tr>
</tbody>
</table>

Percent of test passed

System B is better than system A!!

The problem is with taking the average of ratios

Ratio Game with Percentages (Cont.)

- Both alternatives have the problem of incomparable bases.
- In Alternative 1, the base is the total number of times the experiment is repeated on a system, which is different for the two systems.
- In Alternative 2, the base is sum of repetitions of the two experiments together, which is also different for the two systems.

The Art of Performance Evaluation: Benchmark

to benchmark v. trans. To subject (a system) to a series of tests in order to obtain prearranged results not available on competitive systems

S. Kelly-Bootle
The Devil’s DP Dictionary

- Benchmarking is the process of comparing two systems using standard well known benchmarks.

Misleading by Benchmarking 1

- Different configuration may be used to run the same workload on two systems.
- Different amount of memory, disks ...
- The compilers may be wired to optimize the workload.
- For example, eliminating recognized loops ...
- Test specification may be written so that they are biased towards one machine.
- For example, if the specifications are written based on an existing environment.
- A synchronized job sequence may be used.
- It is possible to manipulate a job sequence so that CPU-bound and I/O-bound steps synchronize to give a better overall performance.

Misleading by Benchmarking 2

- The workload may be arbitrary picked.
- The workload might not be representative of real-world applications.
- Very small benchmarks may be used.
- For example, such small benchmarks can give 100% cache hits, thereby ignoring the inefficiency of memory and cache organization.
- May not show the effect of I/O overhead.
- Few instructions in a loop: By judicious choice of instructions in the loop, the results can be skewed by any amount desired.
- Benchmarks may be manually translated to optimize the performance.
- Often need to manually translated on different systems. The performance may then depend on the ability of the translator than on the system under test.
Summary

- Instruction complexity is only one variable
  - lower instruction count vs. higher CPI / lower clock rate
- Design Principles:
  - simplicity favors regularity
  - smaller is faster
  - good design demands compromise
  - make the common case fast
- Instruction set architecture
  - a very important abstraction indeed!
- Performance measurement - more art than science.