

Performance of Computer Systems

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

These slides are available at:
http://www.csc.lsu.edu/~durrresi/CSC3501_05/



- 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, . . .

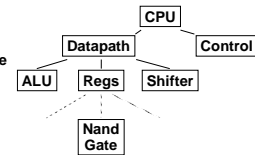
Design Process (cont.)

Design Finishes As Assembly

- Design understood in terms of components and how they have been assembled

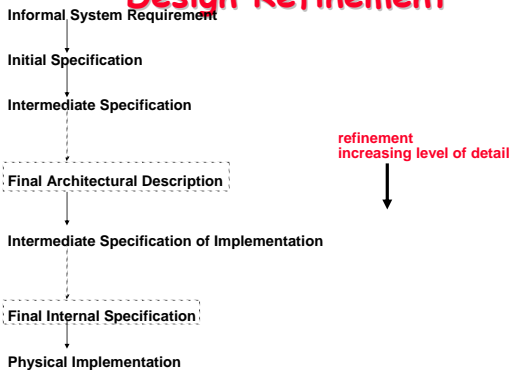
- Top Down *decomposition* of complex functions (behaviors) into more primitive functions

- bottom-up *composition* of primitive building blocks into more complex assemblies

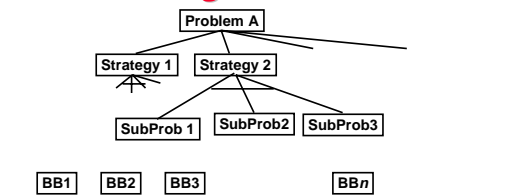


Design is a "creative process," not a simple method

Design Refinement



Design as Search

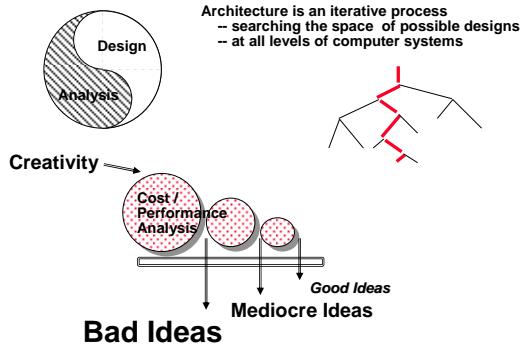


Design involves educated guesses and verification

- 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



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?

Which of these airplanes has the best performance?

Airplane	Passengers	Range (mi)	Speed (mph)	Passenger throughput
Boeing 737-100	101	630	598	228,750
Boeing 747	470	4150	610	286,700
BAC/Sud Concorde	132	4000	1350	178,200
Douglas DC-8-50	146	8720	544	79,429

- How much faster is the Concorde compared to the 747?
- How much bigger is the 747 than the Douglas DC-8?

Computer 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 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.

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,

$$\text{Performance}_X = 1 / \text{Execution time}_X$$

- "X is n times faster than Y"

$$\text{Performance}_X / \text{Performance}_Y = n$$

- Problem:
 - machine A runs a program in 20 seconds
 - machine B runs the same program in 25 seconds

Analysis of CPU Time

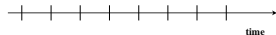
- CPU time depends on the program which is executed, including:
 - a number of instructions executed,
 - types of instructions executed and their frequency of usage.
- Computers are constructed in such way that events in hardware are synchronized using a clock.
- Clock rate is given in Hz (=1/sec).
- 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

$$\text{CPU time} = \frac{\text{seconds}}{\text{program}} = \frac{\text{cycles}}{\text{program}} \times \frac{\text{seconds}}{\text{cycle}}$$

- Clock "ticks" indicate when to start activities (one abstraction):



- 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 $\frac{1}{4 \times 10^9} \times 10^{12} = 250$ picoseconds (ps) cycle time

How to Improve Performance

$$\frac{\text{seconds}}{\text{program}} = \frac{\text{cycles}}{\text{program}} \times \frac{\text{seconds}}{\text{cycle}}$$

- 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?

$$\text{CPU time}_A = \frac{\text{CPU clock cycles}_A}{\text{Clock rate}_A} \quad 10\text{s} = \frac{\text{CPU clock cycles}_A}{4 \times 10^9 \frac{\text{cycles}}{\text{seconds}}}$$

$$\text{CPU clock cycles}_A = 40 \times 10^9 \text{ cycles}$$

$$\text{CPU time}_A = \frac{1.2 \times \text{CPU clock cycles}_A}{\text{Clock rate}_B} \quad \text{Clock rate}_B = 8 \text{ GHz}$$

Measuring Time using Clock Cycles

- CPU execution time for program
= Clock Cycles for a program
× Clock Cycle Time
- One way to define clock cycles:
Clock Cycles for program
= Instructions for a program (called "Instruction Count")
× Average Clock cycles Per Instruction (called "CPI")
- CPI - the average number of clock cycles per instructions is an important parameter
$$CPI = \frac{\text{Clock_cycles_for_a_program}}{\text{Instruction_count}}$$
Instruction_count is the number of instructions executed

Performance Calculation

- CPU execution time for program
= Clock Cycles for program × Clock Cycle Time
- Substituting for clock cycles:
CPU execution time for program
= (Instruction Count × CPI) × Clock Cycle Time
= Instruction Count × CPI × Clock Cycle Time

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

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

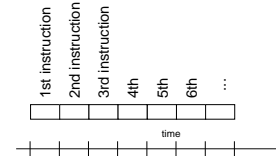
$$\text{CPU time} = \frac{\text{Seconds}}{\text{Program}}$$

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: $\frac{\text{Clock_cycles_for_a_program}}{\text{Instruction_count}}$
 - Calculate: $\frac{\text{Execution Time}}{\text{Clock cycle time}}$ Instruction_count
 - Hardware counter in special register (most CPUs)

How many cycles are required for a program?

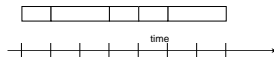
- Could assume that number of cycles equals number of instructions



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



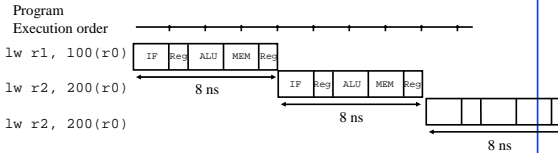
- Multiplication takes more time than addition
- Floating point operations take longer than integer ones
- Accessing memory takes more time than accessing registers
- *Important point: changing the cycle time often changes the number of cycles required for various instructions (more later)*

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;



Every lw instruction needs 8 nsec to execute.

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

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

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.

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_A = I × 2.0 ; CPU clock cycles_B = I × 1.2
CPU time_A = CPU clock cycles_A × CPU clock time_A = I × 2.0 × 250ps = I × 500ps
CPU time_B = CPU clock cycles_B × CPU clock time_B = I × 1.2 × 500ps = I × 600ps

$$\frac{\text{CPU time}_B}{\text{CPU time}_A} = 1.2$$

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

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_i is the average number per instructions for that class.

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?

of Instructions Example

- $CPU \text{ clock cycles}_1 = \sum(CPI_i \times C_i) = (2 \times 1) + (1 \times 2) + (2 \times 3) = 10 \text{ cycles}$
- $CPU \text{ clock cycles}_2 = \sum(CPI_i \times C_i) = (4 \times 1) + (1 \times 2) + (1 \times 3) = 9 \text{ cycles}$
- $CPI_1 = 10/2 = 2$
- $CPI_2 = 9/6 = 1.5$
- 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 and
 - - 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:
 $Clock \text{ cycles for a program} = (x \times 3 + y \times 2 + z \times 4 + w \times 5) = 910 \times 10^6 \text{ clock cycles}$
 $CPU_time = Clock \text{ cycles for a program} / Clock \text{ rate} = 910 \times 10^6 / 500 \times 10^6 = 1.82 \text{ sec}$
- Approach 2:
 $CPI = Clock \text{ cycles for a program} / Instructions \text{ count}$
 $CPI = (x \times 3 + y \times 2 + z \times 4 + w \times 5) / (x + y + z + w) = 3.03 \text{ clock cycles/ instruction}$
 $CPU \text{ time} = Instruction \text{ count} \times CPI / Clock \text{ rate} = (x + y + z + w) \times 3.03 / 500 \times 10^6 = 300 \times 10^6 \times 3.03 / 500 \times 10^6 = 1.82 \text{ sec}$

CPU Time: Example 2

- Consider another implementation of MIPS ISA with 1 GHz clock and
 - - each ALU instruction takes 4 clock cycles,
 - - each branch/jump instruction takes 3 clock cycles,
 - - each sw instruction takes 5 clock cycles,
 - - each lw instruction takes 6 clock cycles.
- Also, consider the same program as in Example 1. Find CPI and CPU time.
- $CPI = (x \times 4 + y \times 3 + z \times 5 + w \times 6) / (x + y + z + w) = 4.03 \text{ clock cycles/ instruction}$
- $CPU \text{ time} = Instruction \text{ count} \times CPI / Clock \text{ rate} = (x + y + z + w) \times 4.03 / 1000 \times 10^6 = 300 \times 10^6 \times 4.03 / 1000 \times 10^6 = 1.21 \text{ sec}$

Analysis of CPU Performance Equation

- $CPU \text{ time} = Instruction \text{ count} \times CPI / Clock \text{ rate}$
- 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

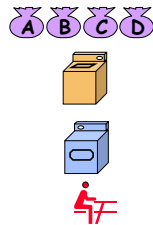
- The table below indicates frequency of all instruction types executed in a "typical" program and, from the reference manual, we are provided with a number of cycles per instruction for each type.

Instruction Type	Frequency	Cycles
ALU instruction	50%	4
Load instruction	30%	5
Store instruction	5%	4
Branch instruction	15%	2

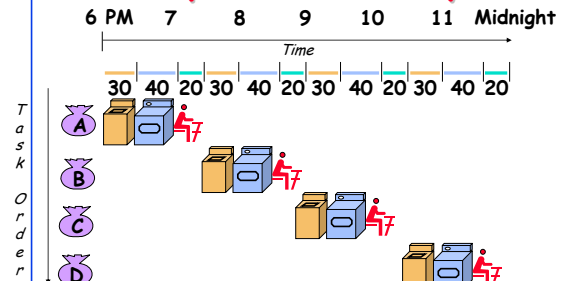
- $CPI = 0.5 \cdot 4 + 0.3 \cdot 5 + 0.05 \cdot 4 + 0.15 \cdot 2 = 4$ cycles/instruction
- The calculation may not be necessary correct since the numbers for cycles per instruction given don't account for pipeline effects.

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

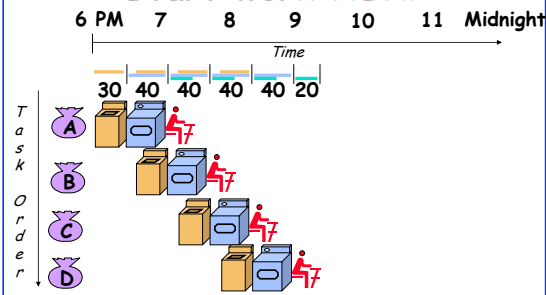


Sequential Laundry



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

Pipelined Laundry



- Pipelined laundry takes 3.5 hours for 4 loads

Pipelining Lessons

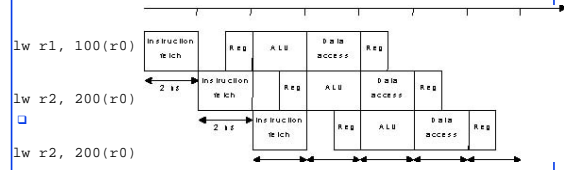
- Pipelining doesn't help latency of single task, it helps throughput of entire workload
- Pipeline rate limited by slowest pipeline stage
- Multiple tasks operating simultaneously
- Potential speedup = Number pipe stages
- Unbalanced lengths of pipe stages reduces speedup
- Time to "fill" pipeline and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



- 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)
- $$\text{MIPS} = \frac{\text{Instruction count}}{\text{Execution time} \times 10^6}$$
- 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

$$\text{Execution time} = \frac{\text{CPU clock cycles}}{\text{Clock rate}}$$

$$\text{CPU clock cycles}_1 = \sum(\text{CPI}_i \times C_i) = ((5 \times 1) + (1 \times 2) + (1 \times 3)) \times 10^9 = 10 \times 10^9$$

$$\text{CPU clock cycles}_2 = \sum(\text{CPI}_i \times C_i) = ((10 \times 1) + (1 \times 2) + (1 \times 3)) \times 10^9 = 15 \times 10^9$$

$$\text{Execution time}_1 = 2.5 \text{ seconds}$$

$$\text{Execution time}_2 = 3.75 \text{ seconds}$$

$$\text{MIPS} = \frac{\text{Instruction count}}{\text{Execution time} \times 10^6}$$

$$\text{MIPS}_1 = 2800$$

$$\text{MIPS}_2 = 3200$$

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*.

$$\text{MFLOPS} = \frac{\text{Number of floating point operations in a program}}{\text{Execution time} \times 10^6}$$

- 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.

Performance Example

- We are interested in two implementations of two similar but still different ISA, one with and one without special real number instructions.
- Both machine 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

Performance Example

- *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 software 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 operations is as follows:
 - - real number multiply needs 30 integer instructions
 - - real number add needs 20 integer instructions
 - - real number divide needs 50 integer instructions

Performance Example

- Consider Program 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 machine.

$$CPI_{MFP} = 0.1 \times 6 + 0.15 \times 4 + 0.05 \times 20 + 0.7 \times 2 = 3.6 \text{ clocks/instr}$$

$$CPI_{MNFP} = 2$$

$$MIPS_{MFP} \text{ rating} = \frac{\text{clock rate}}{CPI \times 10^6} = 270.3$$

$$MIPS_{MNFP} \text{ rating} = 500$$

According to MIPS rating, MNFP is better than MFP!

Performance Example

- B. If Program P on MFP needs 300,000,000 instructions, find time to execute this program on each machine.

	MFP Number of instructions	MNFP Number of instructions
real mul	30×10^6	900×10^6
real add	45×10^6	900×10^6
real div	15×10^6	750×10^6
others	210×10^6	210×10^6
Totals	300×10^6	2760×10^6

- $CPU_time_{MFP} = 300 \times 10^6 \times 3.6 / 1000 \times 10^6 = 1.08 \text{ sec}$
- $CPU_time_{MNFP} = 2760 \times 10^6 \times 2 / 1000 \times 10^6 = 5.52 \text{ sec}$

Performance Example

- C. Calculate MFLOPS for both computers.

$$MFLOPS = \frac{\text{Number of floating point operations in a program}}{\text{Execution time} \times 10^6}$$

$$MFLOPS_{MFP} = 90 \times 10^6 / 1.08 \times 10^6 = 83.3$$

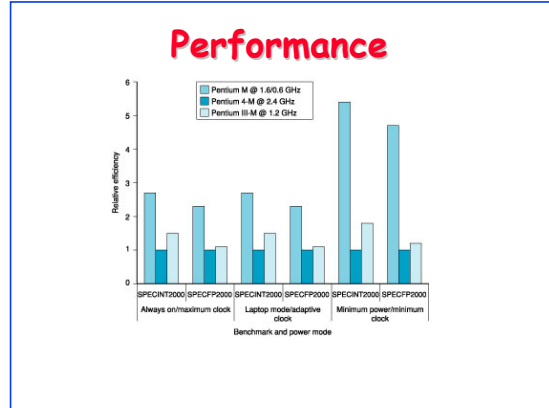
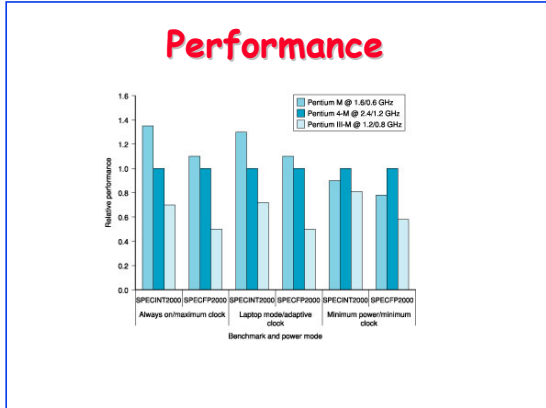
$$MFLOPS_{MNFP} = 90 \times 10^6 / 5.52 \times 10^6 = 16.3$$

Experiment

- 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 Time_i is the execution time for the i th program of a total of n in the workload (benchmark).
- The *weighted arithmetic mean* of execution times is given as:

$$\sum_{i=1}^n \text{Weight}_i \cdot \text{Time}_i$$
 where Weight_i is the frequency of the i th program in the workload.
- The *geometric mean* of execution times is given as:

$$\sqrt[n]{\prod_{i=1}^n \text{Time}_i} \quad \text{where} \quad \prod_{i=1}^n X_i = X_1 \cdot X_2 \cdot X_3 \cdot \dots \cdot X_n$$

Summarizing Performance

SPEC CPU2000 summarizes performance using a geometric mean ratios, with larger numbers indicating higher performance.

CINT2000 is indicator of integer performance and it is given as:

$$\text{CINT2000} = k_1 \times \sqrt[12]{\prod_{i=1}^{12} 1/\text{CPU_time}_i}$$

where k_1 is a coefficient and CPU_time_i is the CPU time for the i th integer program of a total of 12 programs in the workload.

Similarly for floating point performance, CFP2000 is given as:

$$\text{CFP2000} = k_2 \times \sqrt[14]{\prod_{i=1}^{14} 1/\text{FPExecution_time}_i}$$

Geometric mean.

	Time on A		Time on B		Normalized to A		Normalized to B	
	A	B	A	B	A	B	A	B
Program 1	1	10	1	10	0.8	1	1	1
Program 2	1000	100	1	0.1	10	1	1	1
Arithmetic mean of time or normalized time	500.5	55	1	5.05	5.05	1	1	1
Geometric mean of time or normalized time	31.6	31.6	1	1	1	1	1	1

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

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

$$\prod_{i=1}^n a_i \text{ means the product } a_1 \times a_2 \times \dots \times a_n$$

Mean

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

$$\frac{\text{Geometric mean}(X_i)}{\text{Geometric mean}(Y_j)} = \text{Geometric mean}\left(\frac{X_i}{Y_j}\right)$$
- 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 =

Execution Time Unaffected + (Execution Time Affected / Amount of Improvement)

□ 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?

□ Principle: *Make the common case fast*

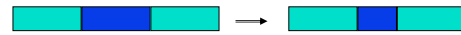
Amdahl's Law

$$ExTime_{new} = ExTime_{old} \times \left[(1 - Fraction_{enhanced}) + \frac{Fraction_{enhanced}}{Speedup_{enhanced}} \right]$$

$$Speedup_{overall} = \frac{ExTime_{old}}{ExTime_{new}} = \frac{1}{(1 - Fraction_{enhanced}) + \frac{Fraction_{enhanced}}{Speedup_{enhanced}}}$$

Best you could ever hope to do:

$$Speedup_{maximum} = \frac{1}{(1 - Fraction_{enhanced})}$$



Example

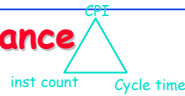
□ Suppose we enhance a machine making all floating-point instructions run five times faster. If the execution time of some benchmark before the floating-point enhancement is 10 seconds, what will the speedup be if half of the 10 seconds is spent executing floating-point instructions?

□ 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

Computer Performance



$$CPU\ time = \frac{Seconds}{Program} = \frac{Instructions}{Program} \times \frac{Cycles}{Instruction} \times \frac{Seconds}{Cycle}$$

	Inst Count	CPI	Clock Rate
Program	X		
Compiler	X	(X)	
Inst. Set.	X	X	
Organization		X	X
Technology			X

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!