Chapter 4 - Performance

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)
Boeing 737-100	101	630	598
Boeing 747	470	4150	610
BAC/Sud Concorde	132	4000	1350
Douglas DC-8-50	146	8720	544

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

Two notions of "performance"

Plane	DC to Paris	Speed	Passengers	Throughput (pmph)
Boeing 747	6.5 hours	610 mph	470	286,700
BAD/Sud Concodre	3 hours	1350 mph	132	178,200

Which has higher performance?

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

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

Response time and throughput often are in opposition

Computer Performance: TIME, TIME, TIME

- **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?**

- ***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**

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"

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

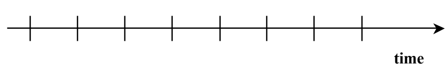
- Problem:
  - machine A runs a program in 20 seconds
  - machine B runs the same program in 25 seconds

Clock Cycles

- Instead of reporting execution time in seconds, we often use cycles

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

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

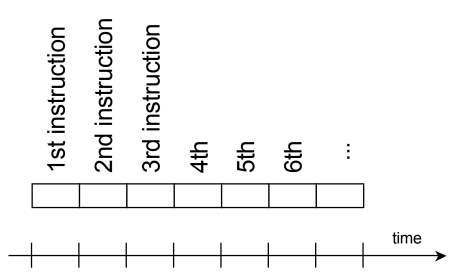
reduce **the # of required cycles for a program, or**

decrease **the clock cycle time or, said another way,**

increase **the clock rate.**

How many cycles are required for a program?

- **Could assume that # of cycles = # of instructions**

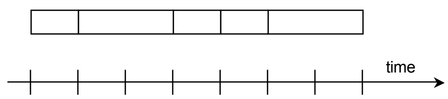


This assumption is incorrect,

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)***

Example

- **Our favorite program runs in 10 seconds on computer A, which has a 400 Mhz. 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?"**

Example

Let C = number of cycles

Execution time = C X clock cycle time = C/ clock rate

On computer A,

C/ 400 MHz = C/ 400 X 10^6 = 10 seconds => C = 400 X 10^7

On computer B, number of cycles = 1.2 X C

What should be B's clock rate so that our favorite program has smaller execution time?

1.2 X C/ clock rate < 10 => 1.2 X 400 X 10^7 / 10 < clock rate

I.e. clock rate > 480 MHz

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

CPI = Average cycles per instruction for the program

Consider a program with 5 instructions

Instruction	#cycles
1	2
2	2
3	4
4	2
5	1
Total	11
CPI	11/5 = 2.2

Another way of saying it is $11 = 5 \times 2.2$
OR CPU cycles = #instructions × CPI

Aspects of CPU Performance

$$\text{cpu time} = \frac{\text{seconds}}{\text{program}} = \frac{\text{instructions}}{\text{program}} \times \frac{\text{cycles}}{\text{instruction}} \times \frac{\text{seconds}}{\text{cycle}}$$

	Instruction Count	CPI	Clock cycle time
Program	X	X	
Compiler	X	X	
Instruction Set	X	X	
Organization		X	X
Technology			X

Program Performance

$$\text{cpu time} = \frac{\text{seconds}}{\text{program}} = \frac{\text{instructions}}{\text{program}} \times \frac{\text{cycles}}{\text{instruction}} \times \frac{\text{seconds}}{\text{cycle}}$$

- **Measuring the components of CPU time for a given program**
  - **Instruction Count**
    - Profiler or simulator
  - **CPI**
    - Depends on hardware implementation *AND* the given program's instruction mix
  - **Clock rate**
    - published

Assignment 2

- **Find the components of CPU time for given program (sort.s) and the program you wrote for Assignment 1 (matrix multiplication program)**
- **You're given**
  - **Clock rate**
  - **#cycles for different categories of instructions (arithmetic, data handling, etc.)**
- **You have to find**
  - **Number of instructions executed by your program**
  - **CPI for your program**
- **Use the profiling technique explained in class**

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

"Average cycles per instruction"

CPI = (CPU Time × Clock Rate) / Instruction Count
= Clock Cycles / Instruction Count

$$\text{CPU time} = \text{Clock cycle time} \times \sum_{j=1}^{n} (CPI_j \times I_j)$$

$$CPI = \sum_{j=1}^{n} CPI_j \times F_j \text{ where } F_j = \frac{I_j}{\text{Instruction Count}}$$

"instruction frequency"

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 10 ns. and a CPI of 2.0
Machine B has a clock cycle time of 20 ns. 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) will always be identical?

CPI Example

For machine A

CPU time = IC × CPI × Clock cycle time

CPU time = IC × 2.0 × 10 ns = 20 IC ns

For machine B

CPU time = IC × 1.2 × 20 ns = 24 IC ns

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

- 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?

$$\text{CPI for sequence } 1 = \frac{2\times1+1\times2+2\times3}{(2+1+2)} = \frac{10}{5}$$

$$\text{CPI for sequence } 2 = \frac{4\times1+1\times2+1\times3}{(4+1+1)} = \frac{9}{6}$$

$$\text{CPU cycles for sequence } 1 = 10/5\times5 = 10$$

$$\text{CPU cycles for sequence } 2 = 9/6\times6 = 9$$

MIPS example

- Two different compilers are being tested for a 100 MHz. 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

- Two different compilers are being tested for a 100 MHz. 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?

$$\text{MIPS} = \text{Millions of instructions per second} = \frac{\text{Number of Instructions}}{\text{Execution time} \times 10^6}$$

$$= \frac{\text{IC}}{\text{IC} \times \text{CPI} \times \text{clock cycle time} \times 10^6} = \frac{\text{IC} \times \text{clock rate}}{\text{IC} \times \text{CPI} \times 10^6} = \frac{\text{clock rate}}{\text{CPI} \times 10^6}$$

For sequence A,

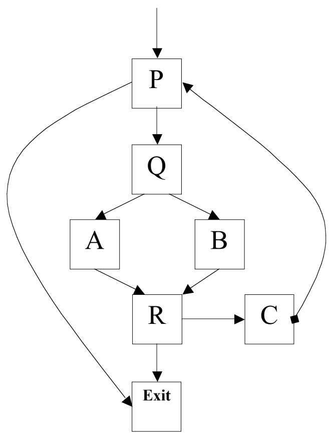
$$\text{CPI} = \frac{(5 \times 1 + 1 \times 2 + 1 \times 3) \times 10^6}{(5 + 1 + 1) \times 10^6} = \frac{10}{7}$$

$$\text{Execution time} = (5 + 1 + 1) \times 10^6 \times \frac{10}{7} \times \frac{1}{100 \times 10^6} = 0.1 \text{ seconds}$$

$$\text{MIPS} = \frac{100 \times 10^6}{10/7 \times 10^6} = 70$$

For sequence B, execution time = 0.15 seconds but MIPS = 80!!!

Profiling a program: Identifying the basic blocks




```
program
    while P do
        if Q then
            A
        else
            B
        fi
        if R then break fi
        C
    od
end
```


Profiling a program

1. Find "basic blocks" of program
2. Create a counter variable for each basic block
3. Insert code that increments the counter for a basic block at the beginning of that block
4. Print out counters at the end of the program
5. Count instructions in each basic block
6. From steps 5 and 6, you have info about the instructions executed by the program

```
                              Saving registers
              sort:    addi $29,$29, -36     # make room on stack for 9 reg
                       sw   $15, 0($29)      # save $15 on stack
                       sw   $16, 4($29)      # save $16 on stack
                       sw   $17, 8($29)      # save $17 on stack
                       sw   $18,12($29)      # save $18 on stack
                       sw   $19,16($29)      # save $19 on stack
                       sw   $20,20($29)      # save $20 on stack
                       sw   $24,24($29)      # save $24 on stack
                       sw   $25,28($29)      # save $25 on stack
                       sw   $31,32($29)      # save $31 on stack
                              Procedure body
Move parameters          move $18, $4        # copy parameter $4 into $18
                         move $20, $5        # copy parameter $5 into $20
Outer loop               add  $19, $0, $0    # i = 0
              for1tst:   slt  $8, $19, $20   # reg $8 = 0 if $19 ≥ $20 (i≥n)
                         beq  $8, $0, exit1  # go to exit1 if $19 ≥ $20 (i≥n)
Inner loop               addi $17, $19, -1   # j = i - 1
              for2tst:   slti $8, $17, 0     # reg $8 = 1 if $17 < 0 (j<0)
                         bne  $8, $0, exit2  # go to exit2 if $17 < 0 (j<0)
                         muli $15, $17, 4    # reg $15 = j * 4
                         add  $16, $18, $15  # reg $16 = v + j
                         lw   $24, 0($16)    # reg $24 = v[j]
                         lw   $25, 4($16)    # reg $25 = v[j+1]
                         slt  $8, $25, $24   # reg $8 = 0 if $25 ≥ $24
                         beq  $8, $0, exit2  # go to exit2 if $25 ≥ $24
Pass parameters          move $4, $18        # 1st parameter of swap is v
and call                 move $5, $17        # 2nd parameter of swap is j
                         jal  swap
Inner loop               addi $17, $17, -1   # j = j - 1
                         j    for2tst        # jump to test of inner loop
Outer loop    exit2:     addi $19, $19, 1    # i = i + 1
                         j    for1tst        # jump to test of outer loop
                              Restoring registers
              exit1:     lw   $15, 0($29)    # restore $15 from stack
                         lw   $16, 4($29)    # restore $16 from stack
                         lw   $17, 8($29)    # restore $17 from stack
                         lw   $18,12($29)    # restore $18 from stack
                         lw   $19,16($29)    # restore $19 from stack
                         lw   $20,20($29)    # restore $20 from stack
                         lw   $24,24($29)    # restore $24 from stack
                         lw   $25,28($29)    # restore $25 from stack
                         lw   $31,32($29)    # restore $31 from stack
                         addi $29,$29, 36    # restore stack pointer
                              Procedure return
                         jr   $31            # return to calling routine
```

GURE 3.21 MIPS assembly version of procedure sort in Figure 3.20 on page 138.

Basis of Evaluation

Pros		Cons
• representative	Actual Target Workload	• very specific • non-portable • difficult to run, or measure • hard to identify cause
• portable • widely used • improvements useful in reality	Full Application Benchmarks	•less representative
• easy to run, early in design cycle	Small “Kernel” Benchmarks	• easy to “fool”
• identify peak capability and potential bottlenecks	Micro benchmarks	• “peak” may be a long way from application performance

Benchmarks

- **Performance best determined by running a real application**
  - **Use programs typical of expected workload**
  - **Or, 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)**
  - **companies have agreed on a set of real program and inputs**
  - **can still be abused (Intel's "other" bug)**
  - **valuable indicator of performance (and compiler technology)**

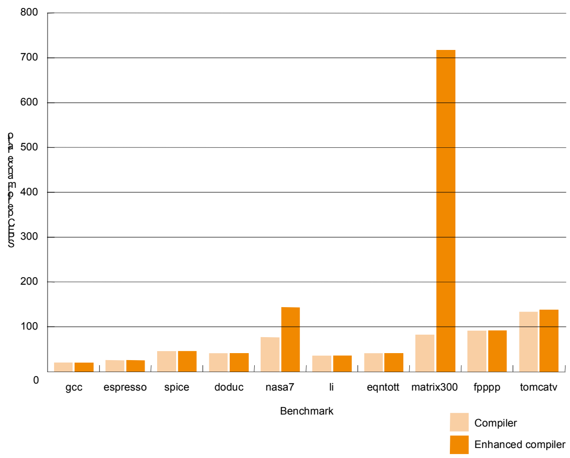
Benchmark Games

- ***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 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…***

  Saturday, January 6, 1996 New York Times

SPEC ‘89

- **Compiler “enhancements” and performance**



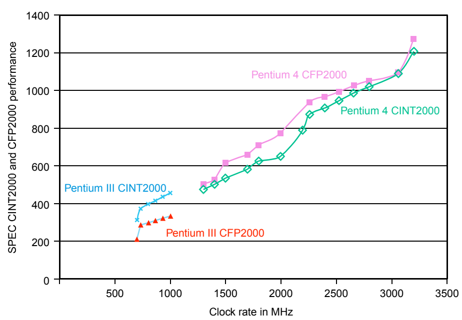
SPEC CPU2000

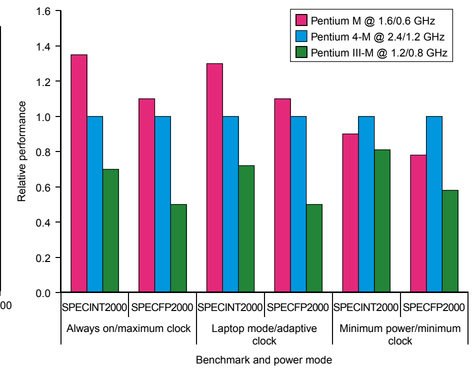
Integer benchmarks		FP benchmarks	
Name	**Description**	**Name**	**Type**
gzip	Compression	wupwise	Quantum chromodynamics
vpr	FPGA circuit placement and routing	swim	Shallow water model
gcc	The Gnu C compiler	mgrid	Multigrid solver in 3-D potential field
mcf	Combinatorial optimization	applu	Parabolic/elliptic partial differential equation
crafty	Chess program	mesa	Three-dimensional graphics library
parser	Word processing program	galgel	Computational fluid dynamics
eon	Computer visualization	art	Image recognition using neural networks
perlbmk	perl application	equake	Seismic wave propagation simulation
gap	Group theory, interpreter	facerec	Image recognition of faces
vortex	Object-oriented database	ammp	Computational chemistry
bzip2	Compression	lucas	Primality testing
twolf	Place and rote simulator	fma3d	Crash simulation using finite-element method
		sixtrack	High-energy nuclear physics accelerator design
		apsi	Meteorology: pollutant distribution

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 2000

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





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***

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