

Benchmarks

Ref: Chapter 2

1

Evaluating Performance

- Performance = 1/Execution Time
- Execution Time of what?
- Ideally we would evaluate computers using the applications we will be running.
 - It's not always feasible.
 - It's nice to be able to *generalize* performance.

2

Revisit Execution Time

$$Time = \frac{Instructions}{program} \times \frac{Cycles}{Instruction} \times \frac{Seconds}{Cycle}$$

The number of instructions depends on the program and the compiler.

The exact choice of instructions depends on the compiler.

3

Compiling

- A Compiler designer selects the set of instructions that should be used to implement some high-level code segment.
- This selection is very important to performance.

4

Example

```
for (i=0;i<100;i++) j=j+i;
```

```
LD R2,j      ; R2 is j
LD R1,#0     ; R1 is i
top:  CMP R1,#100 ; R1 < 100 ?
      BGE bottom ; No - goto bottom
      ADD R1,R2,R2 ; R2 is j
      INC R1      ; increment i
      JMP top     ; go back
bottom: ST R2,j   ; update j
```

↑
Labels

↑
Instructions

↑
Comments

5

Issues

- *Register* allocation (which variables are in which registers and when).
 - registers are *much* faster than memory!
- Choice of instructions
 - different instructions have different CPI.

6

Another possibility

```
for (i=0;i<100;i++) j=j+i;
```

```
LD   R1,0      ; R1 is i
top: CMP  R1,100 ; R1 < 100 ?
     BGE bottom ; No - goto bottom
     LD   R2,j   ; R2 is j
     ADD  R1,R2,R2 ; R2 is j
     ST   R2,j   ; update j
     INC  R1     ; increment i
     JMP  top    ; go back
bottom:
```

7

Comparing Code Segments

- A compiler designer must decide which of 2 possible code segments to use.
- There are 3 *classes* of instructions available:
 - Class A: 1 cycle per instruction
 - Class B: 2 cycles per instruction
 - Class C: 3 cycles per instruction

8

Code Segment Choices

- There are two alternative code segments that can be used:

Code Segment	Instruction Count		
	A	B	C
1	2	1	2
2	4	1	1

9

Code Sequence Comparison

Cycles for CS₁: $2 \times 1 + 1 \times 2 + 2 \times 3 = 10$

Instructions for CS₁: $2 + 1 + 2 = 5$

CPI for CS₁: 2

Cycles for CS₂: $4 \times 1 + 1 \times 2 + 1 \times 3 = 9$

Instructions for CS₂: $4 + 1 + 1 = 6$

CPI for CS₂: 1.5

10

Compiler Writers are underpaid

- Code sequence 2 (CS₂) is much faster even though it includes more instructions!
- In general there are many possible code sequences that can accomplish the same task.
- The job of the compiler designer is important!

11

Picking Test Programs

- The exact programs used for evaluation of performance is important!
- A *benchmark* is a program that is used for testing performance.
 - the *best* benchmarks are real applications!
 - If everyone agreed that Solitaire was the most important program – we could use that as a benchmark.

12

Toy Programs

- Small programs that are specifically designed to be used as benchmarks.
- Example: Homework #1!
- Advantages:
 - easy to isolate the effect of individual operations.
 - easy to develop

13

Problems with toy programs

- It's easy to cheat!
 - processor manufacturer wants his/her processor to look as good as possible.
 - Designs a *special* compiler that recognizes the toy program and generates optimal code.
 - The result is not representative of what the processor will do on other programs.

14

Real Example of Cheating

- A popular benchmark included a program that did matrix multiplication.
- 99% of the *time* is spent on a single line of the program.
- Several companies cheated!

15

matrix300 tuned compiler

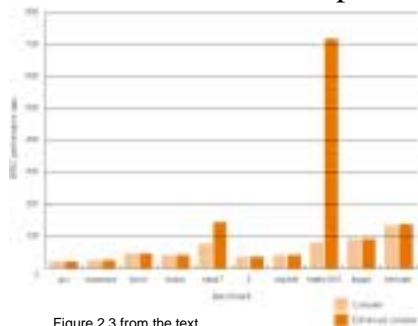


Figure 2.3 from the text.

Other Issues

- The *load* on the machine running the benchmark.
- The size and architecture of the primary memory/cache.

Summarizing Benchmark Results

- Most people don't want to read a large report that details the results of large benchmarks.
- Marketing folks will help us by summarizing the results.
- Marketing folks are wizards of *spin* (don't trust them – read carefully!).

Example: Two Programs

	<u>Computer A</u>	<u>Computer B</u>
Program 1:	1	10
Program 2:	1000	100
Total:	1001	110

A says "I'm 10 times faster than B"

B says "I'm 10 times faster than A"

19

Using Total Execution Time

Computer A takes 1001 sec.

Computer B takes 110 sec.

$$\frac{\text{Performance}_B}{\text{Performance}_A} = \frac{\text{Time}_A}{\text{Time}_B} = \frac{1001}{110} = 9.1$$

B is 9.1 times faster than A

20

Another way to think about it

- Compute (and compare) average execution time as the *arithmetic mean*:

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

21

Normalizing Times

- It is often useful to compare execution times relative to some reference machine.
- Instead of saying it takes 30 seconds, we say it takes “10 times longer on our machine than on a Timex Sinclair”.
- Everybody reports normalized times and we use them to compare performance.

22

Using Arithmetic Mean with Normalized Times

	Time on A	Time on B	Normalized to A		Normalized to B	
			A	B	A	B
Prog. 1	1	10	1	10	0.1	1
Prog. 2	1000	100	1	0.1	10	1
A.M.	500.5	55	1	5.05	5.05	1

23

Geometric Mean

$$GM = \sqrt[n]{\prod_{i=1}^n \text{NormalizeTime}_i}$$

- GM is independent of the choice of machine for normalization.

24

Adding G.M. to our table

	Time on A	Time on B	Normalized to A		Normalized to B	
			A	B	A	B
Prog. 1	1	10	1	10	0.1	1
Prog. 2	1000	100	1	0.1	10	1
A.M.	500.5	55	1	5.05	5.05	1
G.M.	31.6	31.6	1	1	1	1

25

When to use G.M.

- Should always use G.M. when dealing with normalized times (A.M. is not independent of the reference machine).
- BUT: G.M. is not proportional to execution time.
 - This is how we've defined performance!

26

SPEC95 Benchmarks

- Developed by computer companies.
- 8 integer and 10 floating point programs
 - real applications, fixed input.
- All times are normalized to Sun Sparcstation 10/40.
- SPECint95 and SPECfp95 are summary measurements (geometric means).

27

SPECint Programs

Game of GO
Lisp interpreter
Motorola 88K simulator
jpeg compression
Gnu C Compiler - `gcc`
`perl`
`compress`
Database program

28

SPECfp Programs

Mesh Generation
Fluid Flow
Quantum Physics
Astrophysics
Quantum Chemistry
Plasma Physics
Differential Equations
...

29

Improving Performance

- Increase Clock Rate
- Improve Processor Organization
 - Reduce CPI
- Compiler enhancements
 - Reduce Instruction Count
 - Reduce CPI

30

SPECint95 Pentium vs Pentium Pro

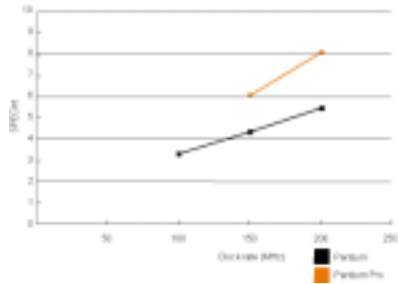


Figure 2.7 from the text.

31

Pentium vs. Pentium Pro

- At same clock rate Pentium Pro is 1.5 times faster than Pentium.
- When clock rate is increased by factor of 2, the increase in performance is by a factor much less than 2!

32

What's up with that?

$$Time = \frac{Instructions}{program} \times \frac{Cycles}{Instruction} \times \frac{Seconds}{Cycle}$$

- According to the above rule, execution time should decrease at same rate clock rate increases.
- The previous graph shows this is not true for the Pentium or Pentium Pro!

33

Oversimplification

- Our rule is too simple (but still a good rule).
- When clock rate increases the speed of main memory is not increased!
- The processor is faster, but the memory is not.
- Later we will study memory architecture and see how to speed memory up...

34

Something to keep in mind

- We can't expect the improvement of just one aspect of a machine to increase performance by a proportional amount.
- We double the clock rate – but don't see performance double.

35

Another Example

- A Program runs in 100 seconds.
 - multiply instructions are responsible for 80 of the 100 seconds.
- We want to speed up the program to run in 60 seconds.
- How much do we need to speed up multiplication?

36

Solution

- We only speed up multiplication, so the new execution time of 60 seconds means:
 - still 20 seconds for other stuff
 - now 40 seconds for multiplication.

Must double speed of multiplication.

37

Five Times Faster

- Same program: 100 seconds total
 - 80 seconds is multiplication.
- We want the program to run 5 times faster.

- How much faster do we need to make multiplication?
 - HINT: Time travel is necessary!

38

MIPS

- Million Instructions Per Second

$$MIPS = \frac{Instruction\ Count}{ExecutionTime\ in\ \mu s}$$

- MIPS is an instruction execution rate

39

The Problem with MIPS

- Does not have anything to do with how much *work* is done.
 - Some instructions do lots of *work*, others do very little.
 - Some instruction sets have lots of complex instructions that do lots of *work*.
 - Some instruction sets have itsy-bitsy instructions – each does very little *work*.

40

More Problems with MIPS

- MIPS varies between programs running on the same computer.
- There is no single MIPS rating for a computer.

41

Even More Problems with MIPS

- MIPS can vary inversely with performance!
 - A program with more instructions can generate a higher MIPS even if it takes longer to run.
 - Check out the example in the book if you don't see how this can happen.

42

Potential Test Questions from Chapter 2

- Performance and Execution Time Equations.
- Clock rate and cycle time.
- What a benchmark is.
- Why MIPS is bad
- Arithmetic vs. Geometric Mean (when and why).
- Exercises 2.13, 2.18, 2.26, 2.44, 2.45

43
