

## Chapter 4

## Important Parameters for System Design

- Performance
- Power consumption
- Cost

## 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?
- How do we evaluate performance?
- 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

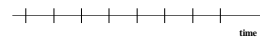
- For some program running on machine X,  
$$\text{Performance}_x = 1 / \text{Execution time}_x$$
- "X is n times faster than Y"  
$$\text{Performance}_x / \text{Performance}_y = n$$
- Problem:
  - machine A runs a program in 20 seconds
  - machine B runs the same program in 25 seconds

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

## How to Improve Performance

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

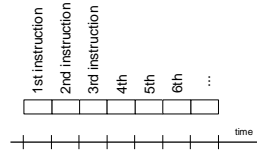
So, to improve performance (everything else being equal) you can either (increase or decrease?)

- \_\_\_? \_\_\_ the # of required cycles for a program, or
- \_\_\_? \_\_\_ the clock cycle time or, said another way,
- \_\_\_? \_\_\_ the clock rate.

©2004 Morgan Kaufmann Publishers 7

## How many cycles are required for a program?

- Could assume that number of cycles equals number of instructions?



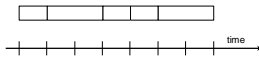
*This assumption is incorrect!!*

*different instructions take different amounts of time on different machines.*

*Why? hint: remember that these are machine instructions, not lines of C code*

©2004 Morgan Kaufmann Publishers 8

## 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)*

©2004 Morgan Kaufmann Publishers 9

## 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?"
- Don't Panic, can easily work this out from basic principles

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

$$\text{Machine A: } 10 \text{ seconds} = \text{Cycles}_A \times 1/400\text{MHz}$$

$$\text{Machine B: } 6 \text{ seconds} = \text{Cycles}_B \times 1/\text{ClockRate}_B$$

$$\text{Cycles}_B = 1.2 \text{ Cycles}_A$$

$$\text{ClockRate}_B = 800\text{MHz}$$

©2004 Morgan Kaufmann Publishers 10

## 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*

©2004 Morgan Kaufmann Publishers 11

## 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.

©2004 Morgan Kaufmann Publishers 12

### CPU Time (or, Execution Time)

$$\frac{\# \text{ of instructions}}{\text{program}} \times \frac{\# \text{ of cycles}}{\text{instruction}} \times \frac{\# \text{ of seconds}}{\text{cycle}}$$

- instruction count  $\times$  CPI  $\times$  cycle time

- instruction count  $\times$  CPI  $\times$   $\frac{1}{\text{clock rate}}$

### 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 10ns and a CPI of 2.0  
Machine B has a clock cycle time of 20ns and a CPI of 1.2

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

$$\text{CPU time}_A = \text{IC} \times \text{CPI} \times \text{cycle time} = \text{IC} \times 2.0 \times 10\text{ns} = 20 \times \text{IC ns}$$

$$\text{CPU time}_B = \text{IC} \times 1.2 \times 20\text{ns} = 24 \times \text{IC ns}$$

So, A is  $(24-20)/20 = 20\%$  faster than B

- 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?

### # 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?**

$$\# \text{ of cycles}_1 = 2 \times 1 + 1 \times 2 + 2 \times 3 = 10$$

$$\# \text{ of cycles}_2 = 4 \times 1 + 1 \times 2 + 1 \times 3 = 7 \text{ So sequence 2 is 30\% faster}$$

$$\text{CPI}_1 = \frac{10}{5} = 2$$

$$\text{CPI}_2 = \frac{7}{6} = 1.167$$

### 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?

$$\# \text{ of instructions} = 5\text{M} + 1\text{M} + 1\text{M} = 7\text{M} \text{ instructions, } \frac{7\text{M}}{100\text{MHz}} = 70\text{ns}$$

$$\# \text{ of cycles} = 5\text{M} \times 1 + 1\text{M} \times 2 + 1\text{M} \times 3 = 8\text{M} \text{ cycles}$$

$$\# \text{ of cycles} = 10\text{M} \times 1 + 1\text{M} \times 2 + 1\text{M} \times 3 = 14\text{M} \text{ cycles} = 140\text{ns}$$

$$\text{So, MIPS}_1 = \frac{7\text{M}}{70\text{ns}} = 100\text{MIPS, MIPS}_2 = \frac{14\text{M}}{140\text{ns}} = 100\text{MIPS}$$

### Benchmarks

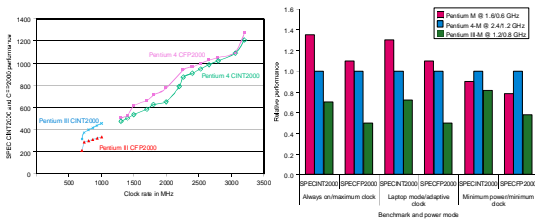
- Performance best determined by running a real application
  - Typical of expected workload
  - Or, typical of expected class of applications e.g., compilers/editors, scientific applications, graphics, etc.
- Small benchmarks
  - nice for architects and designers
  - easy to standardize
  - can be abused
- SPEC (System Performance Evaluation Cooperative)
  - companies have agreed on a set of real program and inputs
  - valuable indicator of performance (and compiler technology)
  - can still be abused

### Standard Performance Evaluation Corporation (SPEC)

- CPU
- Graphics/Application
- HPC/OMP
- Java Client/Server
- Mail Servers
- Network File System
- Web Servers



## SPEC 2000



©2004 Morgan Kaufmann Publishers 25

Execution Time After Improvement =

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

- **Speedup**
  - Execution time (Before) / Execution time (After)
- **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*

©2004 Morgan Kaufmann Publishers 26

## 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 100 seconds, what will the speedup be if half of the 100 seconds is spent executing floating-point instructions?
  - $10/(1+5)$
- 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

$$100 - x + x/5 = 100/3, \quad x = 83.3$$

©2004 Morgan Kaufmann Publishers 27

## 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

©2004 Morgan Kaufmann Publishers 28

## Reading

- [www.spec.org](http://www.spec.org)

©2004 Morgan Kaufmann Publishers 29