September 2

- Performance
- Read 3.1 through 3.4 for Tuesday
- Only 4 classes before 1st Exam!
- Old Fashioned Farmer’s Days

Which of these airplanes has the best performance?

<table>
<thead>
<tr>
<th>Airplane</th>
<th>Passengers</th>
<th>Range(mi)</th>
<th>Speed</th>
<th>Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>777</td>
<td>375</td>
<td>4630</td>
<td>610</td>
<td>228,750</td>
</tr>
<tr>
<td>747</td>
<td>470</td>
<td>4150</td>
<td>610</td>
<td>286,700</td>
</tr>
<tr>
<td>Concorde</td>
<td>132</td>
<td>4000</td>
<td>1350</td>
<td>178,200</td>
</tr>
<tr>
<td>DC-8-50</td>
<td>146</td>
<td>8720</td>
<td>544</td>
<td>79,424</td>
</tr>
</tbody>
</table>

Which communications network is best?

- 56kb modem (56k bits / second, WW)
- Road Runner (1.5M bits / second, WW)
- USB Memory + Sneakers (2G bits / 5 minutes, feet)
- DLT (Digital Linear Tape) + FedEx (1T bit/12 hours)

TIME is THE measure!

- 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?

What kind of time?

- Wall-clock Time
  - counts everything (disk and memory accesses, I/O, etc.)
  - a useful number, but sometimes not good for comparison or analysis 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

DANGER Will Robinson!

- Focus on CPU time can SERIOUSLY distort our world view...
- SYSTEM designers (as opposed to CPU designers) must focus on the USER EXPERIENCE.
“Performance”

- For some program running on machine X,
  \[ \text{Performance}_X = \frac{1}{\text{Execution time}_X} \]
- “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

Cycles

- Instead of reporting execution time in seconds, we often use cycles
  \[
  \begin{align*}
  \text{seconds} & \quad \text{cycles} & \quad \text{seconds} \\
  \text{program} & \quad \text{program} & \quad \text{cycle}
  \end{align*}
  \]
- Clock “ticks” indicate when to start activities (one abstraction):
  - cycle time = time between ticks = seconds per cycle
  - clock rate (frequency) = cycles per second (1 Hertz, 1 cycle/sec)
  A 200 MHz clock has a \( \frac{1}{200 \times 10^6} = 5 \) nanosecond cycle time

How to improve performance?

\[
\begin{array}{ccc}
\text{seconds} & \text{cycles} & \text{seconds} \\
\text{program} & \text{program} & \text{cycle}
\end{array}
\]
So, to improve performance (everything else being equal) you can either

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

How many cycles are required for a program?

- Could assume that \( \# \) of cycles = \( \# \) of instructions

\[
\begin{array}{cccccccc}
\text{Instruction 1} & \text{Instruction 2} & \text{Instruction 3} & \text{Instruction 4} & \text{Instruction 5} \\
\text{second 1} & \text{second 2} & \text{second 3} & \text{second 4} & \text{second 5}
\end{array}
\]

WRONG!
different instructions take different amounts of time on different machines.
WHY?

Instructions take differing numbers of cycles

- Division 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)

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
Do any of these 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 10 ns and a CPI of 2.0
- Machine B has a clock cycle time of 20 ns and a CPI of 1.2

Which 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?

# 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 code 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?

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, 1 million Class B, and 1 million Class C instructions.

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

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

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.
- Synthetic benchmarks (Dhrystone, Whetstone)
  - nice for architects and designers
  - easy to standardize
  - Easy to abuse
- SPEC (System Performance Evaluation Cooperative)
  - companies have agreed on a set of real program and inputs
  - can still be abused
  - valuable indicator of performance (and compiler technology)

SPEC ‘89

- Compiler “enhancements” and performance
SPEC ‘95
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 / Amt of Improvement

\[ T_I = T_U + \frac{T_A}{I} \]

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?

\[ \text{speedup} = \frac{\text{old/new}}{10 / (0.5 \times 10 + 0.5 \times 10/5)} = 1.67 \]

Amdahl’s Law 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?

\[ \frac{100}{4} = 20 + \frac{80}{n} \rightarrow n = 16 \]

How about 5 times faster?

Example
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?

\[ 100/3 = 100f/5 + 100(1-f) \rightarrow f = 5/6 \]
Remember

- Performance is specific to particular programs
  - 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
- Pitfall: expecting improvement in one aspect of a machine’s performance to affect the total performance