Performance?

- Performance measures, report, and summarize
- Make intelligent choices
- See through the marketing hype
- Key to understanding underlying organizational motivation

<table>
<thead>
<tr>
<th>Airplane</th>
<th>Passengers</th>
<th>Range (mi)</th>
<th>Speed (mph)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boeing 737-100</td>
<td>101</td>
<td>630</td>
<td>598</td>
</tr>
<tr>
<td>Boeing 747</td>
<td>470</td>
<td>4150</td>
<td>610</td>
</tr>
<tr>
<td>BAC/Sud Concorde</td>
<td>132</td>
<td>4000</td>
<td>1350</td>
</tr>
<tr>
<td>Douglas DC-8-50</td>
<td>146</td>
<td>8720</td>
<td>544</td>
</tr>
</tbody>
</table>

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

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 and Performance

- 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
- For some program running on machine X,
  - Performance_X = 1 / Execution time_X
- “X is n times faster than Y”
  - Performance_X / 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

  - \(\frac{\text{seconds}}{\text{program}}\) \(\times\) \(\frac{\text{cycles}}{\text{program}}\) \(\times\) \(\frac{\text{seconds}}{\text{cycle}}\)

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

  - \(\frac{1}{\text{cycle rate}} = \frac{\text{cycles per second}}{1 \text{ Hz} = 1 \text{ cycle/sec}}\)

  - A 200 MHz clock has a \(\frac{1}{200 \times 10^9} = 5 \text{ nanoseconds} \times \text{cycle time}\)

- Different instructions take different clock time:
  - Multiplication takes longer than add
  - Floating point takes longer than integer
  - Memory access takes longer than arithmetic or logic

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.
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, MIPS) will always be identical?

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?

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?

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)

Spec 95 programs
Spec 2000 programs

Amdahl's Law

Execution Time After Improvement =

\[
\text{Execution Time Unaffected + \left(\frac{\text{Execution Time Affected}}{\text{Amount of Improvement}}\right)}
\]

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

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

Pitfall: expecting improvement in one aspect of a machine’s performance to affect the total performance

You should not always believe everything you read! Read carefully! (see newspaper articles, e.g., Exercise 2.37)