Computer “Performance”

Readings: 1.6-1.8

BIPS (Billion Instructions Per Second) vs. GHz (Giga Cycles Per Second)

Throughput (jobs/seconds) vs. Latency (time to complete a job)

Measuring “best” in a computer

3.0 GHz

The PowerBook G4 outguns Pentium III-based notebooks by up to 30 percent.*

* Based on Adobe Photoshop tests comparing a 500MHz PowerBook G4 to 850MHz Pentium III-based portable computers

Hyper Pipelined Technology
Performance Example: Homebuilders

<table>
<thead>
<tr>
<th>Builder</th>
<th>Time per House</th>
<th>Houses Per Month</th>
<th>House Options</th>
<th>Dollars Per House</th>
</tr>
</thead>
<tbody>
<tr>
<td>Self-build</td>
<td>24 months</td>
<td>1/24</td>
<td>Infinite</td>
<td>$200,000</td>
</tr>
<tr>
<td>Contractor</td>
<td>3 months</td>
<td>1</td>
<td>100</td>
<td>$400,000</td>
</tr>
<tr>
<td>Prefab</td>
<td>6 months</td>
<td>1,000</td>
<td>1</td>
<td>$250,000</td>
</tr>
</tbody>
</table>

Which is the “best” home builder?
Homeowner on a budget?
Rebuilding Haiti?
Moving to wilds of Alaska?

Which is the “speediest” builder?
Latency: how fast is one house built?
Throughput: how long will it take to build a large number of houses?
Computer Performance

Primary goal: execution time (time from program start to program completion)

\[
Performance = \frac{1}{\text{ExecutionTime}}
\]

To compare machines, we say “X is n times faster than Y”

\[
n = \frac{\text{Performance}_x}{\text{Performance}_y} = \frac{\text{ExecutionTime}_y}{\text{ExecutionTime}_x}
\]

Example: Machine Orange and Grape run a program

Orange takes 5 seconds, Grape takes 10 seconds

Orange is _____ times faster than Grape
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

Example: Unix “time” command

```
linux15.ee.washington.edu> time javac CircuitViewer.java
3.370u 0.570s 0:12.44 31.6%
```

Our focus: user CPU time
  time spent executing the lines of code that are "in" our program
CPU Time

\[
\frac{\text{CPU execution time for a program}}{\text{CPU clock cycles for a program}} = \frac{\text{CPU clock cycles for a program}}{\text{Clock period}} \times \frac{1}{\text{Clock rate}}
\]

Application example:
A program takes 10 seconds on computer Orange, with a 400MHz clock. Our design team is developing a machine Grape with a much higher clock rate, but it will require 1.2 times as many clock cycles. If we want to be able to run the program in 6 second, how fast must the clock rate be?
How do the # of instructions in a program relate to the execution time?

\[
\text{CPU clock cycles for a program} = \text{Instructions for a program} \times \text{Average Clock Cycles per Instruction (CPI)}
\]

\[
\text{CPU execution time for a program} = \text{Instructions for a program} \times \text{CPI} \times \frac{1}{\text{Clock rate}}
\]
CPI Example

Suppose we have two implementations of the same instruction set (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?
Computing CPI

Different types of instructions can take very different amounts of cycles. Memory accesses, integer math, floating point, control flow.

\[
CPI = \sum_{\text{types}} \left( \text{Cycles}_{\text{type}} \times \text{Frequency}_{\text{type}} \right)
\]

<table>
<thead>
<tr>
<th>Instruction Type</th>
<th>Type Cycles</th>
<th>Type Frequency</th>
<th>Cycles * Freq</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU</td>
<td>1</td>
<td>50%</td>
<td></td>
</tr>
<tr>
<td>Load</td>
<td>5</td>
<td>20%</td>
<td></td>
</tr>
<tr>
<td>Store</td>
<td>3</td>
<td>10%</td>
<td></td>
</tr>
<tr>
<td>Branch</td>
<td>2</td>
<td>20%</td>
<td></td>
</tr>
</tbody>
</table>

\[
\text{CPI:}
\]
CPI & Processor Tradeoffs

<table>
<thead>
<tr>
<th>Instruction Type</th>
<th>Type Cycles</th>
<th>Type Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU</td>
<td>1</td>
<td>50%</td>
</tr>
<tr>
<td>Load</td>
<td>5</td>
<td>20%</td>
</tr>
<tr>
<td>Store</td>
<td>3</td>
<td>10%</td>
</tr>
<tr>
<td>Branch</td>
<td>2</td>
<td>20%</td>
</tr>
</tbody>
</table>

How much faster would the machine be if:
1. A data cache reduced the average load time to 2 cycles?

2. Branch prediction shaved a cycle off the branch time?

3. Two ALU instructions could be executed at once?
Warning 1: Amdahl’s Law

The impact of a performance improvement is limited by what is NOT improved:

\[
\text{Execution time after improvement} = \frac{\text{Execution time of unaffected}}{1 + \frac{\text{Execution time affected}}{\text{Amount of improvement}}}
\]

Example: Assume a program runs in 100 seconds on a machine, with multiply responsible for 80 seconds of this time. How much do we have to speed up multiply to make the program run 4 times faster?

5 times faster?
Warning 2: BIPs, GHz ≠ Performance

Higher MHz (clock rate) doesn’t always mean better CPU
Orange computer: 1000 MHz, CPI: 2.5, 1 billion instruction program

Grape computer: 500MHz, CPI: 1.1, 1 billion instruction program

Higher MIPs (million instructions per second) doesn’t always mean better CPU
1 GHz machine, with two different compilers
Compiler A on program X: 10 Billion ALU, 1 Billion Load
Compiler B on program X: 5 Billion ALU, 1Billion Load

Execution Time: A ____  B ____

MIPS: A ____  B ____

<table>
<thead>
<tr>
<th>Instruction Type</th>
<th>Type Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU</td>
<td>1</td>
</tr>
<tr>
<td>Load</td>
<td>5</td>
</tr>
<tr>
<td>Store</td>
<td>3</td>
</tr>
<tr>
<td>Branch</td>
<td>2</td>
</tr>
</tbody>
</table>
Processor Performance Summary

Machine performance:

\[
\text{CPU execution time for a program} = \frac{\text{Instructions for a program} \times \text{CPI} \times \frac{1}{\text{Clock rate}}}{\text{Clock rate}}
\]

Better performance:

_____ number of instructions to implement computations

_____ CPI

_____ Clock rate

Improving performance must balance each constraint

Example: RISC vs. CISC