Why do we worry about performance?

As a consumer:

An application might need a certain level of performance (DOOM) want to be assured that your machine can handle it (here interested in \textit{absolute performance}), or

would like to maximize the use of your dollars (here interested in performance/\$).

As a company selling computer systems:

You compete on the basis of absolute performance, performance per unit price, and increasingly, performance per unit power consumption.

As a designer:

Faced with many design alternatives to meet design objectives. The way to choose the best design alternative is to analyze and measure each one with regards to performance, cost, and power consumption.

How do we characterize computer performance?
Two Notions of “Performance”

<table>
<thead>
<tr>
<th>Plane</th>
<th>DC to Paris</th>
<th>Top Speed</th>
<th>Passengers</th>
<th>Throughput (passengers x mph)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boeing 747</td>
<td>6.5 hours</td>
<td>610 mph</td>
<td>470</td>
<td>286,700</td>
</tr>
<tr>
<td>BAD/Sud Concorde</td>
<td>3 hours</td>
<td>1350 mph</td>
<td>132</td>
<td>178,200</td>
</tr>
</tbody>
</table>

Which has higher performance? Interested in time to deliver 1 passenger? Or, interested in delivering as many passengers per day as possible?

In a computer, time for one task called

- **Response Time** or **Execution Time**

In a computer, tasks per unit time called

- **Throughput** or **Bandwidth**

---

**Definitions**

Performance is characterized as the number of actions per unit time (throughput) or as time per action (response time).

“F is n times faster than S” means:

\[
    n = \frac{\text{performance}(F)}{\text{performance}(S)}
\]

If we are concerned with response time:

\[
    \text{performance}(x) = \frac{1}{\text{execution_time}(x)}
\]

and, “F is n times faster than S” means:

\[
    n = \frac{\text{execution_time}(S)}{\text{execution_time}(F)}
\]
Example of Response Time vs. Throughput

- Time of Concorde vs. Boeing 747?
  - Concord is (6.5 hours / 3 hours) faster
    = 2.2 times faster

- Throughput of Boeing vs. Concorde?
  - Boeing 747: (286,700 pmph / 178,200 pmph)
    = 1.6 times faster

- Boeing is 1.6 times ("60%") faster in terms of throughput

- Concord is 2.2 times ("120%") faster in terms of flying time (response time)

In CS 61C we will focus primarily on response time not throughput.

Confusing Wording on Performance

° We will use “n times faster”; it's less confusing than “m % faster”

° As faster means both increased performance and decreased execution time, to reduce confusion will use “improve performance” or “improve execution time”
What is Time?

- Straightforward definition of time:
  - Total time to complete a task, including disk accesses, memory accesses, I/O activities, operating system overhead, ...
  - "real time", "response time" or "elapsed time"

- Alternative: just time processor (CPU) is working only on your program (since multiple processes running at same time)
  - "CPU execution time" or "CPU time"
  - Often divided into system CPU time (in OS) and user CPU time (in user program)

Measuring performance

- Ideally run typical programs with typical input before purchase, or before even build machine
  - Called a "workload"; For example:
    - Engineer uses compiler, spreadsheet
    - Author uses word processor, drawing program, compression software

- In some situations its hard to do
  - Don’t have access to machine to "benchmark" before purchase
  - Don’t know workload in future
Benchmarks

° Obviously, apparent speed of processor depends on code used to test it

° Need industry standards so that different processors can be fairly compared

° Companies exist that create these benchmarks: “typical” code used to evaluate systems

° Need to be changed every 2 to 5 years since designers could (and do!) target for these standard benchmarks

Example Standardized Benchmarks (1/2)

° Standard Performance Evaluation Corporation (SPEC) SPEC CPU2000
  • SPECINT2000 12 integer (gzip, gcc, crafty, perl, ...)
  • SPECFP2000 14 floating-point (swim, mesa, art, ...)  
    www.spec.org/osg/cpu2000/

° They measure
  - System speed (SPECint2000)
  - System throughput (SPECint_rate2000)

° Benchmarks distributed in source code
° Big Company representatives select workload
  - Sun, HP, IBM, Intel, etc.
### SPEC CPU2000 - SPECfp and SPECint example results

Absolute Values and Values Relative to the IBM eServer pSeries 690Turbo (1.3 GHz CPU).

<table>
<thead>
<tr>
<th>Rank</th>
<th>Machine</th>
<th>SPECfp</th>
<th>SPECint</th>
<th>Test</th>
<th>Relative Values (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>HP Integrity RX2600/Itanium2,1500MHz</td>
<td>2119.0</td>
<td>1322.0</td>
<td>Jun-03</td>
<td>167% 158%</td>
</tr>
<tr>
<td>2</td>
<td>HP Integrity RX5670/Itanium2,1500MHz</td>
<td>2108.0</td>
<td>1312.0</td>
<td>Jun-03</td>
<td>167% 156%</td>
</tr>
<tr>
<td>3</td>
<td>HP ZX6600/Itanium2, 1.5GHz,6MB-L3, H</td>
<td>2106.0</td>
<td>1315.0</td>
<td>Jun-03</td>
<td>166% 157%</td>
</tr>
<tr>
<td>4</td>
<td>SGI Altix 3000 (1.5GHz, Itanium 2)</td>
<td>2100.0</td>
<td>1243.0</td>
<td>Jul-03</td>
<td>166% 148%</td>
</tr>
<tr>
<td>5</td>
<td>NovaScale 4040 Itanium2/1500</td>
<td>2015.0</td>
<td>1107.0</td>
<td>Jul-03</td>
<td>159% 132%</td>
</tr>
<tr>
<td>6</td>
<td>Dell PowerEdge 3250/1.5 GHz Itanium2</td>
<td>1875.0</td>
<td>1099.0</td>
<td>Jul-03</td>
<td>148% 131%</td>
</tr>
<tr>
<td>7</td>
<td>ION Computers 12X2 (1.4GHz Itanium2)</td>
<td>1817.0</td>
<td>926.0</td>
<td>Jul-03</td>
<td>144% 110%</td>
</tr>
<tr>
<td>8</td>
<td>SGI Altix 3000 (1.3GHz, Itanium 2)</td>
<td>1783.0</td>
<td>875.0</td>
<td>Jun-03</td>
<td>141% 104%</td>
</tr>
</tbody>
</table>

### Other Benchmarks

#### ° PCs: Ziff-Davis Benchmark Suite
- Business Winstone is a system-level, application-based benchmark that measures a PC's overall performance when running today's top-selling Windows-based 32-bit applications... it doesn't mimic what these packages do; it runs real applications through a series of scripted activities and uses the time a PC takes to complete those activities to produce its performance scores.
- Also tests for CDs, Content-creation, Audio, 3D graphics, battery life

#### ° Embedded Microprocessors: EEMBC
- Media, automotive, networking, network storage, ... and other related benchmarks
  - [http://www.eembc.org](http://www.eembc.org)
Beyond Benchmarks

° Benchmarks are good tools for empirically measuring computer system performance.

° Need more analysis to understand what factors contribute to performance and how to effect them.

° Start with “clock rate”:
  • Processor manufactures compete with clock-rate, usually in the popular press.
  • Unfortunately it’s not the whole story, in fact sometimes it’s the wrong story.

Clock Rate

° Computers (and other digital systems) are constructed using a clock that runs at a constant rate and determines when events take place in the hardware

  • These discrete time intervals called clock cycles (or informally clocks, cycles, or ticks)
  • Length of clock period: clock cycle time (e.g., 2 nanoseconds or 2 ns) and clock rate (e.g., 500 megahertz, or 500 MHz), which is the inverse of the clock period;
  • Typical clock period these days for high-end processors is <1ns for >1GHz rate.
Measuring Time using Clock Cycles (1/2)

CPU execution time for program

\[ \text{CPU execution time for program} = \text{Clock Cycles for a program} \times \text{Clock Cycle Time} \]

or

\[ \text{CPU execution time for program} = \frac{\text{Clock Cycles for a program}}{\text{Clock Rate}} \]

Measuring Time using Clock Cycles (2/2)

° One way to define clock cycles:

Clock Cycles for program

\[ = \text{Instructions for a program} \times \text{Average Clock cycles Per Instruction} \]

(called “Instruction Count”)

(x abbreviated “CPI”)

° CPI one way to compare two machines with same instruction set, since Instruction Count would be the same
Performance Calculation (1/2)

CPU execution time for program
\[= \text{Clock Cycles for program} \times \text{Clock Cycle Time} \]

Substituting for clock cycles:

CPU execution time for program
\[= (\text{Instruction Count} \times \text{CPI}) \times \text{Clock Cycle Time} \]
\[= \text{Instruction Count} \times \text{CPI} \times \text{Clock Cycle Time} \]

Performance Calculation (2/2)

<table>
<thead>
<tr>
<th>CPU time = Instructions</th>
<th>x Cycles</th>
<th>x Seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program</td>
<td>Instruction</td>
<td>Cycle</td>
</tr>
</tbody>
</table>

\[\text{CPU time} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}\]

Product of all 3 terms: if missing a term, can’t predict time, the real measure of performance
How Calculate the 3 Components?

° **Clock Cycle Time**: in specification of computer (Clock Rate in advertisements)

° **Instruction Count**:
  - Count instructions of small program
  - Use simulator to count instructions
  - Hardware counter in special register
    - (Pentium II,III,4)

° **CPI**:
  - Calculate: \( \frac{\text{Execution Time}}{\text{Instruction Count}} \)
  - Hardware counter in special register (PII,III,4)

Calculating CPI Another Way

° First calculate CPI for each individual instruction (add, sub, and, etc.)

° Next calculate frequency of each individual instruction

° Finally multiply these two for each instruction and add them up to get final CPI (the weighted sum)
Example (RISC processor)

<table>
<thead>
<tr>
<th>Op</th>
<th>Freq(_i)</th>
<th>CPI(_i)</th>
<th>Prod</th>
<th>(% Time)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU</td>
<td>50%</td>
<td>1</td>
<td>.5</td>
<td>(23%)</td>
</tr>
<tr>
<td>Load</td>
<td>20%</td>
<td>5</td>
<td>1.0</td>
<td>(45%)</td>
</tr>
<tr>
<td>Store</td>
<td>10%</td>
<td>3</td>
<td>.3</td>
<td>(14%)</td>
</tr>
<tr>
<td>Branch</td>
<td>20%</td>
<td>2</td>
<td>.4</td>
<td>(18%)</td>
</tr>
</tbody>
</table>

**Instruction Mix**

(Where time spent)

- What if Branch instructions twice as fast?

---

Back to CPU time Formula

\[
\text{CPU time} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}
\]

From the point of view of a computer designer, what factors influence each of these terms?
It’s the Formula that changed the world!

° Computer Architecture Circa 1980: Digital Equipment Corp. VAX

° Very elaborate instruction set:
  OPCODE operand1 operand2 ...
  • Operands can be in memory or registers and accessed through a variety (~20) addressing modes.
  • Any addressing mode could be used for any operand for any opcode.
  • Variable length instruction encoding:
    - Example: integer add: 3-19 bytes

VAX – a CISC (complex instruction set)

° Example addressing modes:
  Register          r4
  Base (displacement)[r4 + offset]
  Immediate         0xffff0101
  Pc-relative        [pc] + offset
  Deferred (indirect) [ [r3 + offset] ]
  Index (scaled)    [r3 + r4*8]
  Auto-increment    (r4)+

° Example opcodes:
  push                push an item onto a stack
  insque              insert an item onto a queue
  aobleq op1 op2      increment op1 and branch if op1=op2
  Special call/ret    handle arguments, stack setup
  poly                take x and a pointer to a set of coefficients, computes a + bx + cx^2 + dx^3+...
RISC (reduced instruction set computer)

Looked like this was going to be the future of computer architecture.... until, armed with “the formula”, Patterson and others argued for RISC.

- Contrast VAX to MIPS:
  - Fixed length instruction encodings
  - One memory addressing mode
  - Memory addressing limited to ld/st
  - Limited opcodes

- Overall the RISC philosophy is to optimize the machine for the commonly used instructions and emulate the others if needed.

CISC implementation issues

\[
\text{CPU time} = \text{Instructions } \times \text{Cycles } \times \text{Seconds}
\]

- Program \quad Instruction \quad Cycle

- The complex instruction set meant lots of complex circuits ⇒ slow clock.

- The complex instructions have high CPI.
  - Inherently many steps
  - Difficult to reduce because of complexity

- The potential for low instruction/program. However, in practice difficult for a compiler to take advantage of all the complexity.

- Therefore, versus RISC, higher sec/cyc, much higher CPI, somewhat lower inst/program not enough to make up the difference ⇒ RISC wins out.

- All new architectures since then have been RISC.
“And in conclusion…” 1/2

° Latency v. Throughput

° Performance doesn’t depend on any single factor: need to know Instruction Count, Clocks Per Instruction and Clock Rate to get valid estimations

CPU time = \( \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}} \)

“And in conclusion…” 2/2

° Benchmarks
  • Attempt to predict performance
  • Updated every few years
  • Measure everything from simulation of desktop graphics programs to battery life

° Megahertz Myth
  • MHz ≠ performance, it’s just one factor