Measuring Performance: Chapter 4!

Or

My computer is faster than your computer...

with thanks to Larry Carter, UCSD
Performance Marches On ...

But what is performance?
Time versus throughput

<table>
<thead>
<tr>
<th>Vehicle</th>
<th>Time to Bay Area</th>
<th>Speed</th>
<th>Passengers</th>
<th>(pm/h)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ferrari</td>
<td>3.1 hours</td>
<td>160 mph</td>
<td>2</td>
<td>320</td>
</tr>
<tr>
<td>Greyhound</td>
<td>7.7 hours</td>
<td>65 mph</td>
<td>60</td>
<td>3900</td>
</tr>
</tbody>
</table>

° Time to do the task from start to finish
  - “execution time”, “latency”, “response time”
° Tasks per unit time
  - “throughput”,

Vehicle: Ferrari, Greyhound
Speed: 160 mph, 65 mph
Time to Bay Area: 3.1 hours, 7.7 hours
Passengers: 2, 60
(pm/h): 320, 3900
Time versus throughput

- **Execution Time** or **Latency** is measured in time.
  - For a SINGLE PROGRAM to execute on a system, usually in a dedicated environment
- **Throughput** is measured in work/time.
  - Total amount of work (instructions, bytes, operations) done by a computer for a given amount of time.
- But “time for one unit of work = 1/throughput” often does not hold
  -- it holds within a bounded region of time
  **pathological examples:**
    - throughput of a computer approaches zero as time goes to infinity (it wears out and stops working)
    - work done by a computer is zero as time goes to zero (not enough time to do a single unit of work)

My farm can grow 8,760 tomatoes in a year; but how long does it take to grow one tomato?

\[
\frac{1}{(8760 \text{ tomatoes/yr})} = 0.00011416 \text{ yrs/tomato} \times 1 \text{ tomato} = 1 \text{ day}!\]
How do you measure Execution Time?

> time foo  
... foo’s results ...
90.7u 12.9s 2:39 65%

- user CPU time? (time CPU spends running your code)
- total CPU time (user + kernel)? (includes op. sys. code)
- Wallclock time? (total elapsed time)
  - Includes time spent waiting for I/O, other users, ...
- Answer depends ...  
  On what you are interested in evaluating!
Cycle: The central “unit of time” on a processor

CPU Time = \#CPU cycles executed \* Cycle time

Cycle Time:
- Every conventional processor has a clock with a fixed cycle time often expressed as a clock rate
  --Rate often measured in GHz = billions of cycles/second
  “I have a 2 GHz machine”
  --Time often measured in ns (nanoseconds)

\[
\text{CYCLE TIME} = \frac{1}{\text{CLOCK RATE}}
\]
Scientific Prefixes:

10^24 (Y) yotta (Greek or Latin octo, "eight")
10^21 (Z) zetta (Latin septem, "seven")
10^18 (E) exa (Greek hex, "six")
10^15 (P) peta (Greek pente, "five")
10^12 (T) tera (Greek teras, "monster")
10^9 (G) giga (Greek gigas, "giant")
10^6 (M) mega (Greek megas, "large")
10^3 (k) kilo (Greek chilioi, "thousand")
10^2 (h) hecto (Greek hekaton, "hundred")
10^1 (da) deka or deca (Greek deka, "ten")
10^-1 (d) deci (Latin decimus, "tenth")
10^-2 (c) centi (Latin centum, "hundred")
10^-3 (m) milli (Latin mille, "thousand")
10^-6 (mu) micro (Latin micro or Greek mikros, "small")
10^-9 (n) nano (Latin nanus or Greek nanos, "dwarf")
10^-12 (p) pico (Spanish pico, "a bit" or Italian piccolo, "small")
10^-15 (f) femto (Danish-Norwegian femten, "fifteen")
10^-18 (a) atto (Danish-Norwegian atten, "eighteen")
10^-21 (z) zepto (Latin septem, "seven")
10^-24 (y) yocto (Greek or Latin octo, "eight")

Usually for Computer Storage

Usually for Computer Time
#Cycles ≠ #Instructions

CPU Time = #CPU cycles executed * Cycle time

#CPU cycles = Instructions executed * CPI

Average Clock Cycles per Instruction

Different codes compile into different numbers of instructions.

for loop
100

Windows OS
5 billion

Each computer design takes a certain amount of time to execute an “average” instruction
## Putting it all together:

One of P&H’s “big pictures”

<table>
<thead>
<tr>
<th>CPU Execution Time</th>
<th>Instruction Count</th>
<th>CPI</th>
<th>Clock Cycle Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note:
- Average CPI is actually hiding some details.

Note:
- Use **dynamic** instruction count (*#instructions executed*), not **static** (*#instructions in compiled code*)
How will I remember?
Re-derive from units

CPU Execution Time = Instruction Count \times CPI \times Clock Cycle Time

What are the units on these measurements?
Dynamic Instruction Count versus Static Instruction Count

```java
int x = 10;
for (int j = 0; j < x; j++)
{
    c[j] = a[j] + b[j];
}

Static IC:
Dynamic IC:
What if x is input?
```

- Static instruction count is determined by the code and the compiler
- Dynamic instruction count is determined by the “choices” made in the execution of the code
  - A video game doesn’t have the same execution time each run...
Practice! $ET = IC \times CPI \times CT$

- gcc runs in 100 sec on a 1 GHz machine
  - How many cycles does it take?

- gcc runs in 75 sec on a 600 MHz machine
  - How many cycles does it take?
How can this possibly be true?

Different IC?
  -> Different ISAs?
  -> Different compilers?

Different CPI?
  -> underlying machine implementation

Different implementation of adders?
  -> for instance, could be pipelined and take multiple cycles
Finding “Average” CPI

- Instruction classes
  - Each take different cycle count
    - Integer operations
    - Floating Point Operations
    - Loads/Stores
    - Multimedia Operations?
  - Can say that “on average” X% of insts from a given class

<table>
<thead>
<tr>
<th>type</th>
<th>Int</th>
<th>FP</th>
<th>MEM</th>
<th>MM</th>
</tr>
</thead>
<tbody>
<tr>
<td># cycles</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>5</td>
</tr>
</tbody>
</table>

|        | 40% | 20% | 35% | 5% |

CPI = \( (0.4 \times 1) + (0.2 \times 4) + (0.35 \times 2) + (0.05 \times \frac{14}{5}) \) = 2.15
Minor Aside from Last Time

- The case of the disappearing MIPS instruction, `bltz`.

The book does not contain all of the MIPS ISA...

MIPS manual posted: check it out

http://www-cse.ucsd.edu/classes/sp07/cse141/docs/
When “Average” CPI fails

- Consider 2 machines with the same clock rate:
  - BigBlue
    - Int 1; FP 4; Mem 2; MM 5
  - SuperVid
    - Int 2; FP 10; Mem 60; MM 1

- Consider 2 compilers for a particular C code:
  - SuperSmart (50$)
    - Int: 10% FP 5% Mem 30% MM 55%
  - GenericSmart (free with machine)
    - Int 50% FP 5% Mem 45% MM 0%

- What is the CPI for each machine with each compiler?
- If you own Big Blue, should you buy the SuperSmart Compiler?
- What if you own SuperVid?
ET = IC * CPI * CT Wrapup

• “Real” CPI exists only:
  - For a particular program with a particular compiler with a particular input.
    • Perhaps a set of common applications (and input sets!)

• You MUST consider all 3 to get accurate ET estimations or machine speed comparisons
  - Instruction Set
  - Compiler
  - Implementation of Instruction Set (386 vs Pentium)
  - Processor Freq (600 Mhz vs 1 GHz)
  - Same high level program with same input
Explaining Execution Time Variation

<table>
<thead>
<tr>
<th>CPU Execution Time</th>
<th>Instruction Count</th>
<th>CPI</th>
<th>Clock Cycle Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Same machine, different programs</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Same program, different machines, but same ISA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Same program, different ISA's</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

which items are likely to be different?
Execution Time? Performance?

• We want higher numbers to be “better”

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

Relative Performance

• “Computer X is r times faster than Y”
or “speedup of X over Y”

\[
\frac{\text{Performance of X}}{\text{Performance of Y}}
\]

we try to avoid saying “X is r times slower …”
what does that mean?
Quick Practice

• Your program runs in 5 minutes on a 1.8 GHz Pentium Pro and in 3 minutes on a 3.2 GHz Pentium 4. How much faster is it on the new machine?

• You get a new compiler for your Pentium 4 from “SmartGuysRUs” which changes the runtime of a different program from Q seconds to B seconds. How much faster is the new program?
How do we achieve increased performance? **(Gene) Amdahl’s Law**

- The impact of an improvement is limited by the fraction of time affected by the improvement.
  - If you make MMX instructions run 10 times as fast, a program which doesn’t use MMX instructions will not run faster.

\[ ET_{\text{new}} = \frac{ET_{\text{old}} \text{ affected/amount of improve}}{ + ET_{\text{old}} \text{ unaffected}} \]

ex: 100 s original: MMX is 50% of run time
ex: 100 s original: MMX is 75% of run time
ex: 100 s original: MMX is 99% of run time

Amdahl → one of the authors on original paper on IBM 360
Amdahl’s Law Practice

- Protein String Matching Code
  - 200 hours ET on current machine, spends 20% of time doing integer instructions
  - How much faster must you make the integer unit to make the code run 10 hours faster?
  - How much faster must you make the integer unit to make the code run 50 hours faster?

A) 1.1
B) 1.25
C) 1.75
D) 2.0
E) 10.0
F) 50.0
G) 1 million times
H) Other
Amdahl’s Law Practice

- **Protein String Matching Code**
  - 4 days ET on current machine
    - 20% of time doing integer instructions
    - 35% percent of time doing I/O
  - Which is the better economic tradeoff?
    - Compiler optimization that reduces number of integer instructions by 25% (assume each integer inst takes the same amount of time)
    - Hardware optimization that makes I/O run 20% faster?
Amdahl’s Law: Last Words

• Corollary for Processor Design:
  - Make the common case fast!
  - Whatever you think the computer will spend the most time doing, spend the most money and the most time making THAT run fast!

• Really: Parallel Processing
  - Only some parts of program can run in parallel
  - Speedup available by running “in parallel” proportional to amount of parallel work available

\[
\text{Speedup}_{\text{max}} = \frac{1}{\text{Serial} + \frac{1-\text{Serial}}{\#\text{processors}}}
\]
Another way of “measuring” performance: Benchmarks

- It’s hard to convince manufacturers to run your program (unless you’re a BIG customer)
- A benchmark is a set of programs that are representative of a class of problems.
  - Microbenchmark - measure one feature of system
    - e.g. memory accesses or communication speed
  - Kernels - most compute-intensive part of applications
    - e.g. Linpack and NAS kernel b’marks (for supercomputers)
  - Full application:
    - SpecInt / SpecFP (int and float) (for Unix workstations)
    - Other suites for databases, web servers, graphics,...
SPEC89 and the compiler

Darker bars show performance with **compiler** improvements (same machine as light bars)

wow
SPEC on Pentium III and Pentium 4

What do you notice?

does Intel cheat? .. or, how could they cheat?
Other SPECs

- HPC (High Performance Computing)
  - Quantum Chemistry, Weather Modeling, Seismic
- JVM (Java)
- JAppletServer
- Web
- Mail
- JBB Java Business Benchmark
- SFS System File Server

Test many things other than the CPU speed - test entire system performance
Comparing Machines w/ Spec benchmarks

Go to Dell website, lookup server machine
(e.g. PowerEdge 1950 w/ Xeon 5160)

Go to Sun website, lookup server machine
(e.g. SunFire X4100 w/ Opteron)

Go to Spec.org and look them up and
compare performance.