Before we start

“What is man in nature?
Nothing in relation to the infinite,
everything in relation to nothing,
a mean between nothing and everything.”

Blaise Pascal, 1670

Outline

- Review
- Measuring and Reporting Performance
- Quantitative Principles of Computer Design
- Things to Remember

Review

- Computing classes: desktop, server, embedd.
- Technology trends
  
<table>
<thead>
<tr>
<th>Capacity</th>
<th>Speed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logic</td>
<td>4x in 3 years</td>
</tr>
<tr>
<td>HDD</td>
<td>4x in 3-4 years</td>
</tr>
<tr>
<td>Disk</td>
<td>4x in 3-4 years</td>
</tr>
</tbody>
</table>

- Cost
  - Learning curve: manufacturing costs decrease over time
  - Volume: the number of chips manufactured
  - Commodity
Review

- Cost of an integrated circuit

\[
IC\ cost = \frac{\text{Die cost} + \text{Testing cost} + \text{Packaging cost}}{\text{Final test yield}}
\]

\[
\text{Cost of die} = \frac{\text{Cost of wafer}}{\text{Dies per wafer} \times \text{Die yield}}
\]

\[
\text{Dies per wafer} = \frac{\pi \times \text{Wafer diameter}^2}{2 \times \text{Die area}}
\]

\[
\text{Die yield} = \frac{\text{Wafer yield} \times \left(1 + \frac{\text{Defects per unit area} \times \text{Die area}}{\alpha}\right)^{-1}}{\text{Total die area}}
\]

Cost-Performance

- Purchasing perspective: from a collection of machines, choose one which has
  - best performance?
  - least cost?
  - best performance/cost?

- Computer designer perspective: faced with design options, select one which has
  - best performance improvement?
  - least cost?
  - best performance/cost?

- Both require: basis for comparison and metric for evaluation

Two “notions” of performance

- Which computer has better performance?
  - User: one which runs a program in less time
  - Computer centre manager: one which completes more jobs in a given time

- Users are interested in reducing Response time or Execution time
  - the time between the start and the completion of an event

- Managers are interested in increasing Throughput or Bandwidth
  - total amount of work done in a given time

An Example

<table>
<thead>
<tr>
<th>Plane</th>
<th>DC to Paris (hour)</th>
<th>Top Speed (mph)</th>
<th>Passengers</th>
<th>Throughput (pas/h)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boeing</td>
<td>6.5</td>
<td>610</td>
<td>470</td>
<td>72</td>
</tr>
<tr>
<td>Concorde</td>
<td>3</td>
<td>1350</td>
<td>132</td>
<td>44 (=132/3)</td>
</tr>
</tbody>
</table>

- Which has higher performance?
  - Time to deliver 1 passenger?
    - Concorde is 6.5/3 = 2.2 times faster (120%)
  - Time to deliver 400 passengers?
    - Boeing is 72/44 = 1.6 times faster (60%)
Definition of Performance

- We are primarily concerned with Response Time.
- Performance [things/sec]
  \[ \text{Performance}(x) = \frac{1}{\text{Execution time}(x)} \]
- "X is n times faster than Y"
  \[ n = \frac{\text{Execution time}(y)}{\text{Performance}(x)} \]
- As faster means both increased performance and decreased execution time, to reduce confusion will use "improve performance" or "improve execution time".

Execution Time and Its Components

- Wall-clock time, response time, elapsed time — the latency to complete a task, including disk accesses, memory accesses, input/output activities, operating system overhead, ...
- CPU time — the time the CPU is computing, excluding I/O or running other programs with multiprogramming — often further divided into user and system CPU times
- User CPU time — the CPU time spent in the program
- System CPU time — the CPU time spent in the operating system

UNIX time command

90.7u 12.9s 2:39 65%
- 90.7 - seconds of user CPU time
- 12.9 - seconds of system CPU time
- 2.39 - elapsed time (159 seconds)
- 65% - percentage of elapsed time that is CPU time
  \[ \frac{90.7 + 12.9}{159} \]

CPU Execution Time

\[ \text{CPU time} = \frac{\text{CPU clock cycles for a program}}{\text{Clock cycle rate}} \]

Definitions
- Instruction count (IC) = Number of instructions executed
- Clock cycles per instruction (CPI)
  \[ \text{CPI} = \frac{\text{CPU clock cycles for a program}}{\text{IC}} \]
- CPI - one way to compare two machines with same instruction set, since Instruction Count would be the same
CPU Execution Time (cont'd)

CPU time = \( IC \times CPI \times \text{Clock cycle time} \)

CPU time = \( \frac{IC \times CPI}{\text{Clock rate}} \)

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

<table>
<thead>
<tr>
<th>Program</th>
<th>IC</th>
<th>CPI</th>
<th>Clock rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compiler</td>
<td>X</td>
<td>0.0</td>
<td></td>
</tr>
<tr>
<td>ISA</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Organisation</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Technology</td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

How to Calculate 3 Components?

- **Clock Cycle Time**
  - In specification of computer
  - (Clock Rate in advertisements)

- **Instruction count**
  - Count instructions in loop of small program
  - Use simulator to count instructions
  - Hardware counter in special register (Pentium II)

- **CPI**
  - Calculate:
    - Execution Time / Clock cycle time / Instruction Count
  - Hardware counter in special register (Pentium II)

Another Way to Calculate CPI

- First calculate CPI for each individual instruction (add, sub, and, etc.): CPI
- Next calculate frequency of each individual instr.: \( \text{Freq} = \frac{IC}{IC} \)
- Finally multiply these two for each instruction and add them up to get final CPI

\[
\text{CPI} = \sum_{i=1}^{n} \left( \frac{IC_i \times CPI_i}{IC} \right)
\]

Choosing Programs to Evaluate Per.

- Ideally run typical programs with typical input before purchase, or before even build machine
  - Engineer uses compiler, spreadsheet
  - Author uses word processor, drawing program, compression software

- **Workload** – mixture of programs and OS commands that users run on a machine

- Few can do this
  - Don’t have access to machine to “benchmark” before purchase
  - Don’t know workload in future
Benchmarks

- Different types of benchmarks
  - Real programs (Ex. MSWord, Excel, Photoshop,...)
  - Kernels - small pieces from real programs (Linpack,...)
  - Toy Benchmarks - short, easy to type and run (Sieve of Eratosthenes, Quicksort, Puzzle,...)
  - Synthetic benchmarks - code that matches frequency of key instructions and operations to real programs (Whetstone, Dhrystone)

- Need industry standards so that different processors can be fairly compared
- Companies exist that create these benchmarks: “typical” code used to evaluate systems

Benchmark Suites

- SPEC - Standard Performance Evaluation Corporation (www.spec.org)
  - originally focusing on CPU performance
    SPEC89/92/95, SPEC CPU2000 (11 Int + 13 FP)
  - graphics benchmarks: SPECviewperf, SPECCapc
  - server benchmark: SPECSFS, SPECWEB
- PC benchmarks (Winbench 99, Business Winstone 99, High-end Winstone 99, CC Winstone 99)
  (www.zdnet.com/etestinglabs/filters/benchmarks)
- Transaction processing benchmarks (www.tpc.org)
- Embedded benchmarks (www.eembc.org)

Comparing and Summarising Performance

- An Example

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>P1 (sec)</td>
<td>20</td>
<td>10</td>
<td>50</td>
</tr>
<tr>
<td>P2 (sec)</td>
<td>100</td>
<td>100</td>
<td>20</td>
</tr>
<tr>
<td>Total (sec)</td>
<td>120</td>
<td>110</td>
<td>70</td>
</tr>
</tbody>
</table>

- What we can learn from these statements?

- We know nothing about relative performance of computers A, B, C!
- One approach to summarise relative performance: use total execution times of programs

Comparing and Sum. Per. (cont’d)

- Arithmetic mean (AM) or weighted AM to track time
  \[ \frac{1}{n} \sum_{i=0}^{n} w_i \times Time_i \]
  Time<sub>i</sub> - execution time for ith program
  w<sub>i</sub> - frequency of that program in workload

- Harmonic mean or weighted harmonic mean of rates tracks execution time
  \[ \frac{\sum_{i=0}^{n} \frac{1}{Rate_i}}{n} \]
  \[ \frac{1}{\sum_{i=0}^{n} \frac{1}{Rate_i}} \]
  Rate<sub>i</sub> - frequency of that program in workload

- Normalized execution time to a reference machine
  - do not take arithmetic mean of normalized execution times, use geometric mean
  \[ \left( \prod_{i=0}^{n} ExTime ratio_i \right)^{\frac{1}{n}} \]
  Problem: GM rewards equally the following improvements:
  Program A: from 2s to 1s, and Program B: from 2000s to 1000s
Quantitative Principles of Design

Where to spend time making improvements?

⇒ Make the Common Case Fast
   - Most important principle of computer design:
     Spend your time on improvements where those improvements will do the most good
     - Example
       • Instruction A represents 5% of execution
       • Instruction B represents 20% of execution
       • Even if you can drive the time for A to 0, the CPU will only be 5% faster

Key questions
- What the frequent case is?
- How much performance can be improved by making that case faster?

Amdahl's Law

Suppose that we make an enhancement to a machine that will improve its performance:

Speedup is ratio:
\[
\text{Speedup} = \frac{\text{ExTime for entire task without enhancement}}{\text{ExTime for entire task using enhancement}}
\]

Amdahl's Law states that the performance improvement that can be gained by a particular enhancement is limited by the amount of time that enhancement can be used.

Computing Speedup

- Fraction_{enhanced} = fraction of execution time in the original machine that can be converted to take advantage of enhancement (E.g., 10/30)
- Speedup_{enhanced} = how much faster the enhanced code will run (E.g., 10/2=5)
- Execution time of enhanced program will be sum of old execution time of the unenhanced part of program and new execution time of the enhanced part of program:
\[
\text{ExTime}_{\text{new}} = \text{ExTime}_{\text{unenhanced}} + \frac{\text{ExTime}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}
\]

Computing Speedup (cont’d)

- Enhanced part of program is Fraction_{enhanced} so times are:
\[
\text{ExTime}_{\text{enhanced}} = \text{ExTime}_{\text{std}} \times (1 - \text{Fraction}_{\text{enhanced}})
\]
- Factor out Time_{std} and divide by Speedup_{enhanced}:
\[
\text{ExTime}_{\text{new}} = \text{ExTime}_{\text{std}} \times \left(1 - \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}} \right)
\]
- Overall speedup is ratio of Time_{old} to Time_{new}:
\[
\text{Speedup} = \frac{1 - \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}}{\text{Speedup}_{\text{enhanced}}}
\]
An Example

- Enhancement runs 10 times faster and it affects 40% of the execution time
- \( \text{Fraction}_\text{enhanced} = 0.40 \)
- \( \text{Speedup}_\text{enhanced} = 10 \)
- \( \text{Speedup}_{\text{overall}} = ? \)

\[
\text{Speedup} = \frac{1}{1 - 0.4 + \frac{0.4}{10}} = \frac{1}{0.64} = 1.56
\]

“Law of Diminishing Returns”

- Suppose that same piece of code can now be enhanced another 10 times
- \( \text{Fraction}_\text{enhanced} = \frac{0.04}{0.60 + 0.04} = 0.0625 \)
- \( \text{Speedup}_\text{enhanced} = 10 \)

\[
\text{Speedup}_{\text{overall}} = \frac{1}{1 - \text{Fraction}_\text{enhanced} + \frac{\text{Fraction}_\text{enhanced}}{\text{Speedup}_\text{enhanced}}} = \frac{1}{0.94 + \frac{0.06}{10}} = 1.059
\]

Using CPU Performance Equations

- Example #1: consider 2 alternatives for conditional branch instructions
  - CPU A: a condition code (CC) is set by a compare instruction and followed by a branch instruction that test CC
  - CPU B: a compare is included in the branch
- Assumptions:
  - on both CPUs, the conditional branch takes 2 clock cycles
  - all other instructions take 1 clock cycle
  - on CPU A, 20% of all instructions executed are cond. branches; since every branch needs a compare, another 20% are compares
  - because CPU A does not have a compare included in the branch, assume its clock cycle time is 1.25 times faster than that of CPU B
- Which CPU is faster?
- Answer the question when CPU A clock cycle time is only 1.1 times faster than that of CPU B

Using CPU Performance Eq. (cont’d)

- Example #1 Solution:
  - CPU A
    - \( \text{CPI}_A = 0.2 \times 2 + 0.8 \times 1 = 1.2 \)
    - \( \text{CPU-time}_A = \text{IC}_A \times \text{CPI}_A \times \text{Clock_cycle_time}_A \)
    - \( \text{IC}_A = 0.8 \times \text{IC}_A \times \text{Clock_cycle_time}_A \times 1.25 \)
  - CPU B
    - \( \text{CPU-time}_B = \text{IC}_B \times \text{CPI}_B \times \text{Clock_cycle_time}_B \)
    - \( \text{Clock_cycle_time}_B = 1.25 \times \text{Clock_cycle_time}_A \)
    - \( \text{IC}_B = 0.8 \times \text{IC}_A \)
    - \( \text{CPI}_B = ? \) compares are not executed in CPU B, so 20%/80%, or 25% of the instructions are now branches
    - \( \text{CPU-time}_B = 0.8 \times \text{IC}_A \times 1.25 \times 1.25 \times \text{Clock_cycle_time}_A \)
  - \( \text{CPU-time}_B / \text{CPU-time}_A = 1.25 \times 1.25 = 1.5625 \Rightarrow \text{CPU A is faster for 4.2%} \)
MIPS as a Measure for Comparing Performance among Computers

- MIPS – Million Instructions Per Second
  
  \[ MIPS = \frac{IC}{CPU\ time \times 10^6} \]

  \[ CPU\ time = \frac{IC \times CPI}{Clock\ rate} \]

- 

  \[ MIPS = \frac{IC}{10^6} \times \frac{1}{CPI \times Clock\ rate} \]

  - Problems with using MIPS as a measure for comparison
    - MIPS is dependent on the instruction set, making it difficult to compare MIPS of computers with different instruction sets
    - MIPS varies between programs on the same computer
    - Most importantly, MIPS can vary inversely to performance
      - Example: MIPS rating of a machine with optional FP hardware
      - Example: Code optimization

MIPS as a Measure for Comparing Performance among Computers (Cont’d)

- Assume we are building optimizing compiler for the load-store machine with following measurements

<table>
<thead>
<tr>
<th>Ins. Type</th>
<th>Freq.</th>
<th>Clock cycle</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU ops</td>
<td>43%</td>
<td>1</td>
</tr>
<tr>
<td>Stores</td>
<td>21%</td>
<td>2</td>
</tr>
<tr>
<td>Branches</td>
<td>24%</td>
<td>2</td>
</tr>
</tbody>
</table>

- Compiler discards 50% of ALU ops
- Clock rate: 500MHz

- Find the MIPS rating for optimized vs. unoptimized code? Discuss it.

- Unoptimized
  - CPI(u) = 0.43 x 1 + 0.57 x 2 = 1.57
  - MIPS(u) = 500MHz/(1.57 x 10^6) = 318.5
  - CPU_time(u) = IC(u) x CPI(u) x Clock_cycle_time
  - IC(u) x 1.57 x 2 x 10^{-9} = 3.14 x 10^{-9} x IC(u)

- Optimized
  - CPI(o) = [(0.43/2) x 1 + 0.57 x 2]/(1 – 0.43/2) = 1.73
  - MIPS(o) = 500MHz/(1.73 x 10^6) = 289.0
  - CPU_time(o) = IC(o) x CPI(o) x Clock_cycle_time
  - 0.785 x IC(u) x 1.73 x 2 x 10^{-9} = 2.72 x 10^{-9} x IC(u)
Things to Remember

- Execution, Latency, Res. time: time to run the task
- Throughput, bandwidth: tasks per day, hour, sec
- User Time
  - time user needs to wait for program to execute: depends heavily on how OS switches between tasks
- CPU Time
  - time spent executing a single program: depends solely on design of processor (datapath, pipelining effectiveness, caches, etc.)

Things to Remember (cont’d)

- Benchmarks: good products created when have good benchmarks
- CPI Law
  \[
  CPU \text{ time} = \frac{\text{Instructions}}{\text{Clock cycles}} \times \frac{\text{Seconds}}{\text{Instructions}} \times \frac{\text{Seconds}}{\text{Clock cycle}}
  \]
- Amdahl’s Law
  \[
  \text{Speedup} = \frac{1}{1 - \frac{\text{Fraction enhanced}}{\text{Speedup enhanced}}}
  \]

Appendix #1

Why not Arithmetic Mean of Normalized Execution Times

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>100</td>
<td>10</td>
<td>20</td>
<td>9</td>
<td>0.1</td>
<td>0.05</td>
<td></td>
</tr>
<tr>
<td>L2 (sec)</td>
<td>1000</td>
<td>100</td>
<td>500</td>
<td>2000</td>
<td>0.1</td>
<td>0.05</td>
<td>0.2</td>
</tr>
<tr>
<td>Total (sec)</td>
<td>10100</td>
<td>1010</td>
<td>550</td>
<td>2000</td>
<td>0.1</td>
<td>0.125</td>
<td></td>
</tr>
<tr>
<td>AM</td>
<td>5050</td>
<td>505</td>
<td>260</td>
<td>1002.5</td>
<td>0.1</td>
<td>0.125</td>
<td></td>
</tr>
<tr>
<td>GM</td>
<td>520</td>
<td>520</td>
<td>260</td>
<td>1002.5</td>
<td>0.1</td>
<td>0.125</td>
<td></td>
</tr>
</tbody>
</table>

AM of normalized execution times: do not use it!
Problem: GM of normalized execution times rewards equally all 3 computers?