Four Questions
The following short quiz consists of 4 questions and tells whether you are qualified to be a "professional". The questions are not that difficult.

1. How do you put a giraffe into a refrigerator?
2. How do you put an elephant into a refrigerator?
3. The Lion King is hosting an animal conference. All the animals attend except one. Which animal does not attend?
4. There is a river you must cross. But it is inhabited by crocodiles. How do you manage it?

Four Answers

• 1. How do you put a giraffe into a refrigerator?
   Answer: Open the refrigerator door, put in the giraffe and close the door. This question tests whether you tend to do simple things in an overly complicated way.

• 2. How do you put an elephant into a refrigerator?
   Wrong Answer: Open the refrigerator door, put in the elephant and close the refrigerator.
   Correct Answer: Open the refrigerator, take out the giraffe, put in the elephant and close the door. This tests your ability to think through the repercussions of your actions.

• 3. The Lion King is hosting an animal conference. All the animals attend except one. Which animal does not attend?
   Answer: The Elephant. The Elephant is in the refrigerator. This tests your memory. OK, even if you did not answer the first three questions correctly, you still have one more chance to show your abilities.

• 4. There is a river you must cross. But it is inhabited by crocodiles. How do you manage it?
   Answer: You swim across. All the Crocodiles are attending the Animal Meeting. This tests whether you learn quickly from your mistakes.

Let’s get mean

<table>
<thead>
<tr>
<th></th>
<th>Arithmetic mean</th>
<th>Arithmetic mean (weighted)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( \frac{1}{n} \sum_{i=1}^{n} \text{time} )</td>
<td>( \sum_{i=1}^{n} \text{weight*time} )</td>
</tr>
<tr>
<td>Program A</td>
<td>100</td>
<td>10</td>
</tr>
<tr>
<td>Program B</td>
<td>30</td>
<td>100</td>
</tr>
<tr>
<td>Total</td>
<td>130</td>
<td>310</td>
</tr>
</tbody>
</table>

What about unequal emphasis of codes in suite?

*Both track total execution time

Administrivia

• Since you’ve decided to stay a while...
  - Check out: www.cse.sc.edu/~kcameron/csse513
  - Add yourself to the csse513 mailing list
  - Get yourself a CSCE Unix account for this class
    - (if you don’t have one)
  - Brush up on your C programming skills
  - Brush up on your UNIX skills
  - Read articles posted on web site
  - Reading list: Chapters 1, 5, 2, A, 3 (in this order)
  - Start Homework #1: 1.3, 1.4, 1.6, 1.8, 1.10, 1.16

Weighted arithmetic mean example

<table>
<thead>
<tr>
<th></th>
<th>Comp A</th>
<th>Comp B</th>
<th>Comp C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program 1</td>
<td>1</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>Program 2</td>
<td>1000</td>
<td>100</td>
<td>20</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>W_{1}=5, W_{2}=5</th>
<th>W_{1}=909, W_{2}=091</th>
<th>W_{1}=999, W_{2}=001</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computer A</td>
<td>500.5</td>
<td>91.909</td>
<td>1.999</td>
</tr>
<tr>
<td>Computer B</td>
<td>55</td>
<td>18.10</td>
<td>10.09</td>
</tr>
<tr>
<td>Computer C</td>
<td>20</td>
<td>20</td>
<td>20</td>
</tr>
</tbody>
</table>
Be normal

Normalized execution time: normalize to a particular machine by dividing all execution times by chosen machine's time.

Example: Program P1 has the following execution times:
- On machine A: 10 secs
- On machine B: 100 secs
- On machine C: 150 secs

Normalized to A: A=1, B=10, C=15
Normalized to B: A=.1, B=1, C=1.5

Normalized example

\[
\frac{1}{n} \sum_i \text{Execution time ratio} \quad \sqrt[n]{\prod_i \text{Execution time ratio}}
\]

<table>
<thead>
<tr>
<th>Normalized to A</th>
<th>Normalized to B</th>
<th>Normalized to C</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETR-P1</td>
<td>1 10 20</td>
<td>1 1 1 2 .05 5 1</td>
</tr>
<tr>
<td>ETR-P2</td>
<td>1 1 .02 10</td>
<td>1 1 2 50 5 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Normalized example</th>
<th>Normalized to A</th>
<th>Normalized to B</th>
<th>Normalized to C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fast rate</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NGM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ETR-P1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ETR-P2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Normalized example</th>
<th>Normalized to A</th>
<th>Normalized to B</th>
<th>Normalized to C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A B C A B C A B C</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>A B C A B C A B C</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Getting Meaner

Taking the average of the normalized times

\[
\frac{1}{n} \sum_i \text{Execution time ratio} \quad \sqrt[n]{\prod_i \text{Execution time ratio}}
\]

<table>
<thead>
<tr>
<th>Comp A</th>
<th>Comp B</th>
<th>Comp C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preg 1</td>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>Preg 2</td>
<td>1000</td>
<td>100</td>
</tr>
</tbody>
</table>

Geometric vs. Arithmetic

- Arithmetic mean
  - Provides weighted average
  - Pros: proportional to overall execution time
  - Cons:
    - Can be rigged easily (disproportionate problem size)
    - Cannot use with normalizing
- Normalized Geometric mean
  - Provides relative performance of machines to ref
  - Pros: same results regardless of ref machine
  - Cons:
    - Not proportional to overall execution time
    - Large % change in small overall time contributor can skew
- Suggestions
  - Weight programs according to their actual frequency
  - Use problem size to pre-normalize program execution time
  - Combine approaches: summary of simple means and relative performance to base machine

Principals of Arch Design

- Make common case fast (90/10 Rule)
- Amdahl's Law
  - Law of diminishing returns
- Speedup
  - Achieved performance improvement over original

\[
\text{Speedup} = \frac{\text{performance}_{\text{new}}}{\text{performance}_{\text{old}}} \quad \text{executiontime}_{\text{new}} = \text{executiontime}_{\text{old}} + \text{executiontime}_{\alpha}
\]

Amdahl's Law

Execution time of any code has two portions
- Portion I: not affected by enhancement
- Portion II: affected by enhancement

\[
\text{executiontime}_{\text{new}} = \text{executiontime}_{\text{old}} + \text{executiontime}_{\alpha}
\]

\[
\alpha \text{ is } \% \text{ of original code that would benefit from enhancement}
\]

As $n \to \infty$, execution time$_{new} = (1-\alpha) \cdot$ execution time$_{old}$

\[
\text{execution time}_{\alpha} = (1-\alpha) \cdot \text{execution time}_{\text{old}} + \alpha \cdot \text{execution time}_{\text{new}}
\]

n is speedup factor of old/new execution times for portion II
Amdahl’s Law

\[ \text{execution time}_{\text{new}} = (1 - \alpha) \times \text{execution time}_{\text{old}} + \alpha \times \text{new time execution} \]

Example: \( \alpha = 80\% \)

\[ \text{Speedup} = \frac{\text{execution time}_{\text{old}}}{\text{execution time}_{\text{new}}} = \frac{1}{1 - \alpha + \alpha} \]

\( \alpha = \frac{1}{20} = 0.05 \)

\[ \text{Speedup} = \frac{1}{1 - 0.05} = 2.15 \]

Example

- Enhancement: Vector mode
- Portions of code containing computations run 20x faster in vector mode.
- What % of original code must be vectorizable to achieve \( \text{speedup}_{\text{overall}} = 2 \)?

\[ \text{Speedup}_{\text{overall}} = \frac{\text{execution time}_{\text{old}}}{\text{execution time}_{\text{new}}} = \frac{1}{1 - \alpha + \alpha} \]

\( \alpha = \frac{1}{5} = 0.2 \)

CPU Performance Equation

\[ \text{CPU time} = \frac{\text{cycles}}{\text{clock rate}} \times \text{cycles per program} \]

\[ \text{CPU time} = \frac{\text{instructions}}{\text{cycles per instruction}} \times \text{instructions per program} \]

Influenced by

- ISA/compiler
- Organization/ISA
- Hardware/organization

Example

- All ops work on registers
- Assume 25% ALU ops use operand not used again
- Add ALU ops with one source in memory (cycle count = 2)
- Extended ISA increases branches by 1 cycle
- What is CPI of this design?
Example

<table>
<thead>
<tr>
<th>Instruction Type</th>
<th>Frequency</th>
<th>Clock cycle count</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU ops</td>
<td>43%</td>
<td></td>
</tr>
<tr>
<td>Loads</td>
<td>21%</td>
<td></td>
</tr>
<tr>
<td>Stores</td>
<td>12%</td>
<td></td>
</tr>
<tr>
<td>Branches</td>
<td>24%</td>
<td></td>
</tr>
</tbody>
</table>

% instr count before enhancement

25% ALU ops use operand not used again

Normalized total

\[ \sum_{i=1}^{n} CPI \times \text{frequency} = (1\times0.36) + (2\times0.124) + (2\times0.112) + (2\times0.134) + (3\times0.27) = 1.91 \]

Which is faster?

\[ \text{cpu time} = \frac{\text{instructions}}{\text{cycles}} \times \frac{\text{seconds}}{\text{cycles}} \]

\[ \text{cpu time} = \text{instructions} \times \frac{1.91}{\text{clockrate}} \]

Fallacies and Pitfalls

- Two processors, same ISA, judge by one benchmark
- Application areas may differ
- Benchmarks remain valid indefinitely
- Peak performance tracks observed performance
- Optimize without considering implementation
- As complexity increases time to market decreases
- Synthetic benchmarks predict real performance
- MIPS/MFLOPS is an accurate measure for comparing performance
- Comparing hand-coded assembly to compiler optimized
- Neglecting the cost of software in cost/performance
- Falling prey to Amdahl's Law

Speed vs. Time

\[ \text{cpu time} = \frac{\text{instructions}}{\text{cycles}} \times \frac{\text{seconds}}{\text{cycles}} \]

\[ \text{cpu time} = \frac{\text{instructions}}{\text{cycles}} \times \frac{1.91}{\text{clockrate}} \]

\[ \text{cpu time} = 1.70 + \text{instructions} \times \frac{1}{\text{clockrate}} \]