

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.

Four Answers

- 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 you 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.

Administrivia

- Since you've decided to stay a while...
 - Check out: www.cse.sc.edu/~kcameron/csce513
 - Add yourself to the csce513 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

Let's get mean

Arithmetic mean

$$\frac{1}{n} \sum_{i=1}^n time$$

Arithmetic mean (weighted)

$$\sum_{i=1}^n weight * time$$

	Computer 1	Computer 2	Computer 3
Program A	100	10	1
Program B	30	300	200
Total	130	310	201

What about unequal emphasis of codes in suite?

*Both track total execution time

Weighted arithmetic mean example

Arithmetic mean (weighted)

$$\sum_{i=1}^n weight * time$$

	Comp A	Comp B	Comp C
Prog 1	1	10	20
Prog 2	1000	100	20

	$W_1=.5, W_2=.5$	$W_1=.909, W_2=.091$	$W_1=.999, W_2=.001$
Computer A	500.5	91.909	1.999
Computer B	55	18.10	10.09
Computer C	20	20	20

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

Execution time ratio

Getting Meaner

Taking the average of the normalized times

Normalized arithmetic mean

Normalized geometric mean

$$\frac{1}{n} \sum_{i=1}^n \text{Execution time ratio}$$

$$\sqrt[n]{\prod_{i=1}^n \text{Execution time ratio}}$$

	Comp A	Comp B	Comp C
Prog 1	1	10	20
Prog 2	1000	100	20

Normalized example

$$\frac{1}{n} \sum_{i=1}^n \text{Execution time ratio}$$

$$\sqrt[n]{\prod_{i=1}^n \text{Execution time ratio}}$$

	Normalized to A			Normalized to B			Normalized to C		
	A	B	C	A	B	C	A	B	C
ETR P1	1	10	20	1	1	2	.05	.5	1
ETR P2	1	.1	.02	10	1	.2	50	5	1

Bad idea	NAM	Normalized to A			Normalized to B			Normalized to C		
		A	B	C	A	B	C	A	B	C
	NGM	1	1	.63	1	1	.63	1.58	1.58	1

Geometric vs. Arithmetic

- Arithmetic mean
 - Provides weighted average
 - Pros: proportional to overall execution time
 - Cons:
 - Cannot use with normalizing
 - Can be rigged easily (disproportionate problem size)
- 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

$$\text{Speedup}_{\text{max}} = \frac{\text{performance}_{\text{new}}}{\text{performance}_{\text{old}}} = \frac{\text{execution time}_{\text{old}}}{\text{execution time}_{\text{new}}}$$

Amdahl's Law

Execution time of any code has two portions

Portion I: not affected by enhancement

Portion II: affected by enhancement

$$\text{execution time}_{\text{old}} = \text{execution time}_{p1} + \text{execution time}_{p2}$$

α is % of original code that would benefit from enhancement

$$\text{As } n \rightarrow \text{infinity, execution time}_{\text{new}} \rightarrow (1-\alpha) * \text{execution time}_{\text{old}}$$

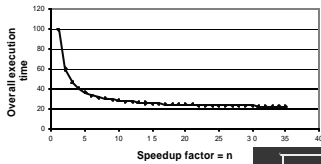
$$\text{execution time}_{\text{new}} = (1-\alpha) * \text{execution time}_{\text{old}} + (\alpha) * \frac{\text{execution time}_{\text{old}}}{n}$$

n is speedup factor of old/new execution times for portion II

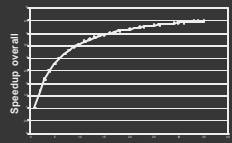
Amdahl's Law

$$\text{executiontime}_{\text{new}} = (1 - a) * \text{executiontime}_{\text{old}} + (a) * \frac{\text{executiontime}_{\text{old}}}{n}$$

Example: alpha = 80%



$$\text{Speedup}_{\text{max}} = \frac{\text{executiontime}_{\text{old}}}{\text{executiontime}_{\text{new}}} = \frac{1}{(1 - a) + \frac{a}{n}}$$



Example

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

$$\text{Speedup}_{\text{max}} = \frac{\text{executiontime}_{\text{old}}}{\text{executiontime}_{\text{new}}} = \frac{1}{(1 - a) + \frac{a}{n}}$$

$$2 = \frac{1}{(1 - a) + \frac{a}{20}} \quad a = .5263$$

Example

- Enhanced mode is used 50% of resulting execution time_{new}
- Portions of code using enhanced mode improve performance by factor 10x
- What is speedup for fast mode?

First find α (% of code affected by enhancement)

$$\text{execution time}_{\text{new}} = \underbrace{(0.5) * \text{execution time}_{\text{old}}}_{\text{not enhanced}} + \underbrace{(0.5) * \text{execution time}_{\text{old}}}_{\text{enhanced}}$$

$$(1 - a) * \text{executiontime}_{\text{old}} = \frac{a * \text{executiontime}_{\text{old}}}{10}$$

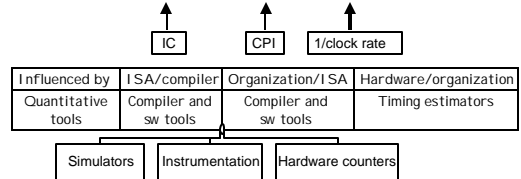
$$a = .90909$$

$$\text{Speedup}_{\text{max}} = \frac{1}{(1 - a) + \frac{a}{n}} = \frac{1}{(1 - 0.9) + \frac{0.9}{10}} = 5.3$$

CPU Performance Equation

$$\text{cpu time} = \frac{\text{total \# clock cycles per program (cycles)}}{\text{clock rate (MHz = } 10^6 \text{ cycles/sec)}}$$

$$\text{cpu time} = \frac{\text{instructions}}{\text{program}} * \frac{\text{cycles}}{\text{instructions}} * \frac{\text{seconds}}{\text{cycle}} = \frac{\text{seconds}}{\text{program}}$$



The CPI formula

$$\text{CPI} = \sum_{i=1}^n \text{CPI}_i * \left(\frac{\text{IC}_i}{\text{Instruction count}} \right) = \sum_{i=1}^n \text{CPI}_i * \text{frequency}$$

$$\text{CPI} = \text{Pipeline CPI} + \text{MemorySystemCPI}$$

Example

Register-register ALU:
Load \$r1, @mem1
Add \$r0, \$r2, \$r1

Register-memory ALU:
Add \$r0, \$r2, @mem1

- 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?

Instruction type	Frequency	Clock cycle count
ALU ops	43%	1
Loads	21%	2
Stores	12%	2
Branches	24%	2

Example

Instruction type	Frequency	Clock cycle count
ALU ops	43%	1
Loads	21%	2
Stores	12%	2
Branches	24%	2

% instr count before enhancement

ALU ops 43%
Loads 21%
Stores 12%
Branches 24%

25% ALU ops use operand not used again

r-r: $\frac{3}{4} * 43\% = 32\%$
r-m: $\frac{1}{4} * 43\% = 11\%$
Loads: 21-11%=10%
Stores: 12%
Branches: 24%

Normalized total

r-r: 32/89=36%
r-m: 11/89=12.4%
Loads: 10/89=11.2%
Stores: 12/89=13.4%
Branches: 24/89=27%

Total number of instructions reduced by 11%

$$CPI_i = \sum_{i=1}^n CPI_i * frequency = (1 * .43) + (2 * .21) + (2 * .12) + (2 * .24) = 1.57$$

$$CPI_i = \sum_{i=1}^n CPI_i * frequency = (1 * .36) + (2 * .124) + (2 * .112) + (2 * .134) + (3 * .27) = 1.91$$

Speed vs. Time

Which is faster?

$$CPI_i = \sum_{i=1}^n CPI_i * frequency = (1 * .43) + (2 * .21) + (2 * .12) + (2 * .24) = 1.57$$

$$CPI_i = \sum_{i=1}^n CPI_i * frequency = (1 * .36) + (2 * .124) + (2 * .112) + (2 * .134) + (3 * .27) = 1.91$$

$$cpu\ time = \frac{instructions}{program} * \frac{cycles}{instructions} * \frac{seconds}{cycle} = \frac{seconds}{program}$$

$$cpu\ time = instructions * 1.91 * \frac{1}{clockrate}$$

$$cpu\ time = (.89 * instructions) * 1.91 * \frac{1}{clockrate}$$

$$cpu\ time = 1.70 * instructions * \frac{1}{clockrate}$$

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