Lesson 2

HPCA Metrics and Evaluation

This lesson shows methods for measuring improvements to a system. These methods are important to the computer architect during the decision making process because they show which designs will yield the best performance.

Performance usually refers to the speed of the processor. Speed can be broken down into two aspects of speed:

Can measure performance as either:

- **<u>Latency</u>:** how much time does it take to go from the start to the finish (END to END)
- **<u>Throughput</u>:** how many tasks can be completed in one time unit.

Throughput is NOT always equal to 1/Latency (b/c of pipelining or multiple "execution units")

Comparing Performance

Performance can be compared using speedup. speedup is a ratio of times!

Speedup is defined as "X is N times faster than Y".

<u>Speedup</u>: N = Speed(X) / Speed(Y) = Throughput(X) / Throughput(Y)
= Latency(Y) / Latency(X)

*Note the difference when using Latency

Speedup < 1 : The performance has gotten worse with the newer version.
The degradation is because of worse performance. (higher latency or lower throughput)

Speedup > 1: The performance has gotten better with the newer version.
The improvement is through shorter execution time (lower latency)
or with Higher Throughput.

~~Performance is proportional to 1/latency~~

Performance is proportional to the Throughput
or
Performance is proportional to $\frac{1}{\text{Latency}}$

Measuring Performance

<u>Performance</u> ~ 1/Execution time

"Execution Time" = "Latency"

When comparing performances of different processors, a standard task should be used. This will lead to a standard execution time for the given task for the processor.

Benchmarks

Benchmarks are standard tasks for measuring processor performance. A benchmark is a suite of programs that represent common tasks.

Types of Benchmarks

<u>Real Applications</u> - most realistic, $\underbrace{\text{most difficult to set up}}_{}$ - used for real machine comparisons.

need an OS, peripherals, drivers, etc.

<u>Kernels</u> - most time consuming part of an application, is still difficult to set up - used for testing prototypes.

for example, what if we don't have a compiler for this machine yet?

Synthetic - similar to kernels, but simpler to compile, used for design studies of potential new machines
Peak performance - used for marketing. - "in theory, how many instructions per second should this machine be able to execute"

Benchmark Standards
There are benchmark organizations that create the standard benchmarks.

Summarizing Performance
Summarize performance using the average execution time.
If Speedup is compared to Speedup; the Geometric mean, NOT the average, must be used.

summarized speedup = GeoMean (all speedups)
or
= speedup($\frac{\text{GeoMean } X}{\text{GeoMean } Y}$)

Geometric mean = (Product of the terms)$^{1/\text{Number of terms}}$

	Comp X	Comp Y	Speedup
App A	9 s	18 s	2
App B	10 s	7 s	0.7
App C	5 s	11 s	2.2
Average	8 s	12 s	X

AVG (8+12) ≠ AVG(2, 0.7, 2.2)

CANNOT use average for speedup, MUST use Geometric Mean

Iron Law of Performance
The Iron Law of Performance is :
CPU Time = # of Instructions in the Program * Cycles per Instructions * Clock cycle Time

$= \left(\frac{\text{\# of instructions}}{\text{Program}}\right) \cdot \left(\frac{\text{\# of clock cycles}}{\text{1 instruction}}\right) \cdot \left(\frac{\text{\# of seconds}}{\text{1 clock cycle}}\right) = \frac{\text{\# of seconds}}{\text{Program}}$

All three aspects are important in decision making in computer architecture

(w) GeoMean CompA $= (9 \cdot 10 \cdot 5)^{1/3} = 7.66$
(x) GeoMean CompB $= (18 \cdot 7 \cdot 11)^{1/3} = 11.149$
(y) GeoMean speedup $= (2 \cdot 0.7 \cdot 2.2)^{1/3} = 1.455$
(z) speedup of (w) over (x) $= \frac{11.149}{7.66} = 1.455$

so (y) + (z) produce the same result

- # of Instructions in the Program: is effected by the algorithm, the compiler used, and/or the instruction set used.
- Cycles per Instructions: is affected by the instruction set and/or the processor design
- Clock cycle Time: is affected by processor design, circuit design, and/or transistor physics
- Computer architects influence the instruction set and the processor design.

The Iron Law for Unequal Instruction Times
CPU Time = [Sum of(Inst/Program * cycles/Inst)] * Time/cycle

$= \sum_x \left(\frac{\text{\# of instructions}_x}{\text{Program}} \cdot \underbrace{\frac{\text{\# of cycles}}{\text{1 instruction}_x}}_{CPI_x} \right) \cdot \frac{\text{\# of seconds}}{\text{1 clock cycle}}$

1 billion = 1×10^9
giga = 10^9

Amdahl's Law
Used for measuring overall speedup when only a fraction of the system is improved.
Speedup = 1/((1-Frac of Enhancement) + (Frac Enhancement/Speedup Enhancement))

% of original execution time NOT affected by enhancement

Calculate this based on the improvement on total execution time

Frac of Enhancement = % of original execution time that is affected by enhancement (speedup)

(original = before the improvement)

NOT the % of the instructions in the program

Amdahl's Law Implications
Use Amdahl's law to determine which design decisions will yield the best system.
Make the Common Case Fast : small improvements for a large percentage of the system are better than large improvements on a small percentage of the system.

Lhadma's Law ("Amdahl" backwards)
While trying to make the common case fast, do not make the uncommon case worse.

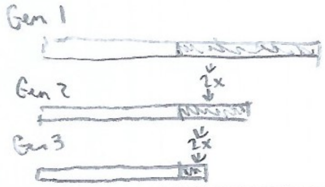
Diminishing Returns

Example:
- improve 90% of program by 2x
- slow down the rest (10%) by 10x

$$\text{speedup} = \frac{1}{\frac{0.1}{0.1} + \frac{0.9}{2}} = 0.7 \leftarrow \text{overall slowdown}$$

Diminishing Returns

Once the easy changes for improvement have been made, further improvements will not yield great returns. More and more work must be done to obtain small improvements and these improvements will result in small to nearly zero speedups.



the more improvements to the shaded part, the less % of the execution time it affects (so no longer the "common case")

Amdahl's Law Example:

- Program w/ 50 billion instructions
- Processor @ 2 GHz
- Improved branch instruction CPI from $4 \rightarrow 2$

What is the overall speedup from this improvement?

Program Breakdown

Instruction	% of # of instructions in Program	CPI
INT	40%	1
BRANCH	20%	$4 \rightarrow 2$
LOAD	30%	2
STORE	10%	3

$$\text{Speedup} = \frac{1}{(1 - FRAC_{ENH}) + \left(\frac{FRAC_{ENH}}{Speedup_{ENH}}\right)}$$

Original execution time $= (40\% \cdot 1 + 20\% \cdot 4 + 30\% \cdot 2 + 10\% \cdot 3) \cdot \underbrace{50}_{\text{50 billion}} \cdot 10^9 \times \frac{1}{\underbrace{2 \cdot 10^9}_{2\text{GHz}}} = \underline{52.5 \text{ seconds}}$

New execution time $= (40\% \cdot 1 + 20\% \cdot \underline{\underline{2}} + 30\% \cdot 2 + 10\% \cdot 3) \cdot 50 \cdot 10^9 \times \frac{1}{2 \cdot 10^9} = \underline{42.5}$

$Speedup_{ENH} = \frac{4}{2} = 2$

$FRAC_{ENH}$ = % of ORIGINAL execution time that was enhanced

\# of branch instructions $= (50 \cdot 10^9)(20\%) = \underline{10 \cdot 10^9}$ branch instructions in program

$$FRAC_{ENH} = \frac{(10 \cdot 10^9 \text{ branch instructions})\left(\frac{4 \text{ cycles}}{1 \text{ branch instruction}}\right)\left(\frac{1 \text{ second}}{2 \cdot 10^9 \text{ cycles}}\right)}{52.5 \text{ seconds}} = 0.380952381 \approx \underline{38.0952\%}$$

$$\text{Speedup} = \frac{1}{(1 - 38.0952\%) + \left(\frac{38.0952\%}{2}\right)} = \underline{1.2353 \approx 1.24}$$

We could also just calculate the speedup w/ the old → new execution times:

$$\frac{52.5}{42.5} = 1.2353 \approx 1.24$$

matches ✓