Performance of Computers

Reference:

David A. Patterson and John L. Hennessy, Computer Organization and Design: The Hardware/Software Interface, Morgan Kaufmann / Elsevier, Fifth Edition, 2014.

Performance in General

"X is n times faster than Y" means:

$$n = \frac{Performance(X)}{Performance(Y)}$$

Basic Performance Metrics

- **Response time(Execution Time):** the time between the start and the completion of a task (in time units)
- **Throughput:** the total amount of tasks done in a given time period (in number of tasks per unit of time)

Example: Car assembly factory:

4 hours to produce a car (response time)

6 cars per an hour produced (throughput)

Computer Performance: Introduction

- The computer user is interested in response time (or execution time) – the time between the start and completion of a given task (program).
- In many real computer systems, changing either execution time or throughput often affects the other.

- Performance = 1/Execution Time

Problem:

- machine A runs a program in 20 seconds.
- machine B runs the same program in 25 seconds.
- how many times faster is machine A?

$$\frac{Performance(A)}{Performance(B)} = \frac{Executiontime(B)}{Executiontime(A)} = n$$

n = 1.25

CPU Time or CPU Execution Time

- **CPU execution time:** Total time a CPU spends computing on a given task (excludes time for I/O or running other programs). This is also referred to as simply CPU time.
- Computers are constructed is such way that events in hardware are synchronized using a clock.
- Clock rate is given in Hz (=1/sec).
- A clock rate defines durations of discrete time intervals called clock cycle times or clock cycle periods:
- clock_cycle_time = 1/clock_rate (in sec)

Computing CPU Time

The time to execute a given program can be computed as

CPU time = CPU clock cycles for a program * clock cycle time

CPU time = CPU clock cycles for a program / clock rate

- Clock cycles for a program is a total number of clock cycles needed to execute all instructions of a given program.

CPU time = Instruction count * CPI / Clock rate

- Instruction count is a number of instructions executed, sometimes referred as the instruction path length.
- CPI – the average number of clock cycles per instruction (for a given execution of a given program)
- CPI = Clock cycles for a program / Instructions count

Calculating CPI

The table below indicates frequency of all instruction types executed in a “typical” program.

Instruction Type	Frequency	Cycles
ALU Instruction	50%	4
Load Instruction	30%	5
Store Instruction	5%	4
Branch Instruction	15%	2

CPI = 0.5*4 + 0.3*5 + 0.05*4 + 0.15*2 = 4 cycles/instruction

CPU Time: Example 1

- Consider an implementation of MIPS ISA with 500 MHz clock and
  - each ALU instruction takes 3 clock cycles,
  - each branch/jump instruction takes 2 clock cycles,
  - each sw instruction takes 4 clock cycles,
  - each lw instruction takes 5 clock cycles.

Also, consider a program that during its execution executes:

- x=200 million ALU instructions
- y=55 million branch/jump instructions
- z=25 million sw instructions
- w=20 million lw instructions

- Find CPU time. Assume sequentially executing CPU.

Approach 1:

Clock cycles for a program = (x*3 + y*2 + z*4 + w*5)

$= 910 * 10^6$ clock cycles

CPU time = Clock cycles for a program / Clock rate

$=910 * 10^6 / 500 * 10^6 = 1.82$ sec

Approach 2:

CPI = Clock cycles for a program / Instructions count

CPI = (x*3 + y*2 + z*4 + w*5)/ (x + y + z + w)

= 3.03 clock cycles/ instruction

CPU time = Instruction count * CPI / Clock rate

$=(x+y+z+w) * 3.03 / 500 * 10^6$

$= 300 * 10^6 * 3.03 /500 * 10^6$

=1.82 sec

CPU Time Example

- CPU clock rate is 500 MHz
- Program takes 45 million cycles to execute
- What's the CPU time
- = 0.09 seconds

CPI Example

- Suppose we have two implementations of the same instruction set architecture (ISA).

For some program,

Machine A has a clock cycle time of 10 ns. and a CPI of 2.0

Machine B has a clock cycle time of 20 ns. and a CPI of 1.2

- Which machine is faster for this program, and by how much?

Assume that # of instructions in the program is 1,000,000,000.

CPU $Time_A = 10^9 * 2.0 * 10 * 10^{-9} = 20$ seconds

CPU $Time_B = 10^9 * 1.2 * 20 * 10^{-9} = 24$ seconds

n= Perf (A)/Perf(B)=cpu time(B)/cputime(A)=24/20= 1.2 times.

So Machine A is faster.

Components that affect CPU performance

	Instruction Count	CPI	Clock rate
Algorithm	X	X	
Programming Language	X	X	
Compiler	X	X	
ISA	X	X	X

CPU time = Instruction count * CPI / Clock rate

Analysis of CPU Performance Equation

CPU time = Instruction count * CPI / Clock rate

How to improve (i.e. decrease) CPU time:

- Clock rate: hardware technology & organization,
- CPI: organization, ISA and compiler technology,
- Instruction count: ISA & compiler technology.

Many potential performance improvement techniques primarily improve one component with small or predictable impact on the other two.

Basic Measurement Metrics

- Comparing Machines

  Metrics:

  Execution time

  Throughput

  CPU time

  MIPS –million instructions per second

  MFLOPS –million floating point operations per second

- Comparing Machines Using Sets of Programs

  Arithmetic mean, weighted arithmetic mean.

MIPS – Million Instructions Per Second

MIPS = Instruction count / (CPU time * $10^{6)}$

CPU time = Instruction count * CPI / clock rate

MIPS = clock rate / (CPI * 10^{6})

- For example, a program that executes 3 million instructions in 2 seconds has a MIPS rating of 1.5

MIPS – Million Instructions Per Second

- The problems with MIPS rating as a performance measure:
  - difficult to compare computers with different instruction sets.
  - MIPS varies between programs on the same computer.

MIPS Example

Two different compilers are being tested for a 500 MHz. machine with three different classes of instructions .

Class	A	B	C
CPI	1	2	3
Compiler-1	5 Billion instr	1	1
Compiler-2	10	1	1

- Which sequence will be faster according to execution time?

CPU time = Instr.count * CPI / Clock rate.

CPU time-1 = $(5 * 1 + 1 * 2 + 1 * 3) * 10^9 / (500 * 10^6) = 20$ seconds

CPU time-2 = $(10 * 1 + 1 * 2 + 1 * 3) * 10^9 / (500 * 10^6) = 30$ seconds

MIPS Example

- Which sequence will be faster according to MIPS?

MIPS = Instruction count / CPU time * 10^6

MIPS-1 = (5 + 1 + 1) * 10^9 / 20 * 10^6 = 350

MIPS-2 = (10 + 1 + 1) * 10^9 / 30 * 10^6 = 400

Amdahl’s Law

- Amdahl's law is an expression used to find the maximum expected improvement to an overall system when only part of the system is improved.

- Architecture design is very bottleneck-driven –make the common case fast, do not waste resources on a component that has little impact on overall performance

Amdahl's Law

Speedup = Execution time before improvement / Execution time after improvement.

Assumption: The Execution time before is 1 for some unit of time.

Speedup = 1/ ((1-fraction enhanced) +(fraction enhanced/factor of improvement))

- Let Speedup be denoted by "S", fraction enhanced be denoted by "f_E", and factor of improvement be denoted by "f_I". Then we can write the above equation as

$$S = ((1 - f_E) + (f_E / f_I))^{-1}$$

Amdahl's Law

- What Kinds of Problems Do We Solve with Amdahl's Law?

The three problem types are as follows:

1.Determine S given f_E and f_I

2. Determine f_I given S and f_E

3. Determine f_E given S and f_I

Amdahl's Law

Predict System Speedup:

- Let a program have 40 percent of its code enhanced (so $f_E = 0.4$) to run 2.3 times faster (so $f_I = 2.3$). What is the overall system speedup S? .

$$S = ((1 - f_E) + (f_E / f_I))^{-1}$$

$$= 1.292$$

Predict Speedup of Fraction Enhanced

Let a program have 40 percent of its code enhanced (so $f_E = 0.4$) to yield a system speedup 4.3 times faster (so $S = 4.3$). What is the factor of improvement f_I of the portion enhanced?

Case #1:

Can we do this? In other words, let's determine if by enhancing 40 percent of the system, it is possible to make the system go 4.3 times faster …

Assume the limit, where f_I = infinity,

$$S = ((1 - f_E) + (f_E / f_I))^{-1} = 1 / (1 - f_E) = ((1 - 0.4))^{-1}$$
$$= 1.67$$

So S = 1.67 is the **maximum possible speedup, and we cannot achieve S = 4.3 !!**

Predict Speedup of Fraction Enhanced

Case #2:

A different case: Let's determine if by enhancing 40 percent of the system, it is possible to make the system go 1.3 times faster …

Assume the limit, where f_I = infinity,

$$S = ((1 – f_E) + (f_E / f_I))^{-1} = 1 / (1 – f_E) = ((1 – 0.4))^{-1} = 1.67$$

So S = 1.67 is the **maximum possible speedup, and we can achieve S = 1.3 !! .**

$$f_I = f_E \cdot (1/S – (1 – f_E))^{-1} = 2.367$$

Predict Fraction of System to be Enhanced

Let a program have a portion f_E of its code enhanced to run 4 times faster (so $f_I = 4$), to yield a system speedup 3.3 times faster (so S = 3.3). What is the fraction enhanced (f_E)?

Can this be done? Assuming f_I = infinity, $S = 3.3 = ((1 - f_E))^{-1}$

so minimum $f_E = 0.697$

Yes, this can be done for maximum f_I, so let's solve the equation to determine actual f_E

$$S = ((1 - f_E) + (f_E / f_I))^{-1}$$

$f_E = 0.697 / 0.75 =$ **0.929**

Example

1. Overall speedup if we make 90% of a program run 10 times faster.

   $f_E = 0.9$ $f_I = 10$ $s = ?$

   $s = 5.26$

2. Overall speedup if we make 80% of a program run 20% faster

   $f_E = 0.8$ $f_I = 1.2$ $s=?$

   $s = 1.153$

Example

- Assume a new web-server with a CPU being 10 times faster on computation than the previous web-server. I/O performance is not improved compared to the old machine. The web-server spends 40% of its time in computation and 60% in I/O. How much faster is the new machine overall?

S= ?, $f_E = 0.4$, $f_I = 10$

S = 1.56

A common transformation required in graphics processors is square root. Implementations of floating-point (FP) square root vary significantly in performance, especially among processors designed for graphics.

Suppose FP square root (FPSQR) is responsible for 20% of the execution time of a critical graphics benchmark. One proposal is to enhance the FPSQR hardware and speed up this operation by a factor of 10.

The other alternative is just to try to make all FP instructions in the graphics processor run faster by a factor of 1.6; FP instructions are responsible for half of the execution time for the application. The design team believes that they can make all FP instructions run 1.6 times faster with the same effort as required for the fast square root. Compare these two design alternatives.

Sol.

$$\text{Speedup}_{\text{FPSQR}} = \frac{1}{(1 - 0.2) + \frac{0.2}{10}} = \frac{1}{0.82} = 1.22$$

$$\text{Speedup}_{\text{FP}} = \frac{1}{(1 - 0.5) + \frac{0.5}{1.6}} = \frac{1}{0.8125} = 1.23$$