Computer components and performance

Response time and throughput

If you were running a program on two different desktop computers, you can say that the faster one is the desktop computer that gets the job done first. If you were running a datacenter have several servers (server can be understood as a computer) running jobs submitted by many users, you can say that the faster computer was the one that completed the most jobs during a day. As an individual computer user, you are interested in reducing **response time** (the time between the start and completion of a task, or *execution time*). On the other hand, datacenter managers are often interested in increasing **throughput** (somewhere called *bandwidth*, the total amount of work done in a given time, in physics we called frequency but we changed the number of motions with the number of jobs).

Time

> Time is the measure of computer performance: the computer that performs the same amount of work in the least time is the fastest.

We called the total time to complete a task, including disk accesses, memory accesses, input/output (I/O) activities, operating system overhead - everything (the time between the start and completion of a task) is **response time**, or **elapsed time** and we have formula

$$t_{resp} = t_{end} - t_{start} = \text{CPU time} + \text{IO wait time}$$

In **elapsed time**, we have a smaller time which is **CPU time** (or *CPU execution time*). **CPU time** is just only time spent processing a given job (Discounts I/O time, other jobs shares, it's actual time the CPU spends computing for a specific task). **CPU time** comprises user CPU time and system CPU time.

Improve performance

We can improve performance by improve response time (t_{resp}) or throughput

But, if you use **faster CPU** $\implies t_{resp}$ reduced $\implies$ throughput increased because time for jobs decreased. Hence, in this case, both of response time and throughput are improve

Another case, we can add more CPUs. In this case, throughput improved because in one time more jobs are done at the same time, so response time ***may be*** improve

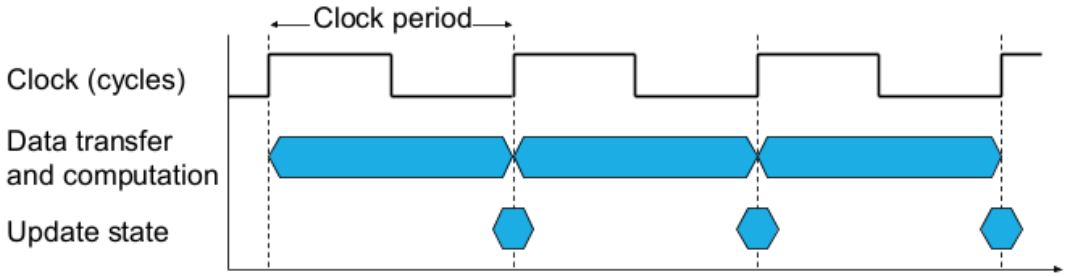
Example: In computer, if you want to do some jobs (Like 5 jobs), CPU maybe process each a job. For example, if CPU process job 1, 4 jobs left in queue, and if we have more CPU, we can decrease jobs in queue. So, we can decrease IO time because we do not need more time for transferring jobs from queue to CPU

Key term

- Response time (execution time): The total time required for the computer to complete a task, including disk accesses, memory accesses, I/O activities, operating system overhead, CPU execution time, and so on.
- Throughput (bandwidth): Another measure of performance, it is the number of tasks completed per unit time.
- CPU time: It is actual time the CPU spends computing for a specific task (Discounts I/O time, other jobs shares)

CPU Clocking

As mentioned above, computer users we need to care about time (t_{resp}) but we need new unit convenient to think about performance in other metrics. Almost all computers are constructed using a clock that determines when events take place in the hardware. These discrete time intervals are called **clock cycles** (or *ticks, clock ticks, clock periods, clocks, cycles*). (Designers refer to the length of a clock period both as the time for **a complete clock cycle** (e.g., 250ps) and as **the clock rate** (e.g., 4 GHz), which is the inverse of the clock period)



- Clock period (clock cycles): duration of a clock cycle

  e.g., $250ps = 0.25ns = 250 \times 10^{-12}s$
- Clock frequency (clock rate): cycles per second
  e.g., $4.0GHz = \frac{1}{0.25}ns = 4000MHz = 4.0 \times 10^{9}Hz$

CPU Time

Users and designers often examine performance using different metrics. If we could relate these different metrics, we could determine the effect of a design change on the performance as experienced by the user (Above we said users only need to focus to time response). Since we are confining ourselves to CPU performance at this point, the bottom-line performance measure is CPU execution time. So, we have simple formula relates the most basic metrics (clock cycles and clock cycle time) to CPU time:

$$\text{CPU execution time}_{\text{ for a program}} = \text{CPU clock cycle}_{\text{ for a program}} \times \text{Clock cycle time}$$

Alternatively, because clock rate and clock cycle time are inverses,

$$\text{CPU execution time}_{\text{ for a program}} = \frac{\text{CPU clock cycle}_{\text{ for a program}}}{\text{Clock rate}}$$

Note: Above formula makes it clear that the hardware designer can improve performance by reducing the number of **clock cycles** required for a program or the length of the clock cycle

Instruction count and CPI

As we all know, each program comes with a multiple command (Actually, from your code, compiler generated **instructions** to execute, and the computer - CPU, just only know how to run **instructions**, so CPU execution time equal to the sum of time run per instruction) so CPU execution time depends on the number of **instructions**. But, in CPU execution time's formula above did not include any reference to the number of instructions needed for the program. So, one way to think about execution time is that it equals the number of instructions executed multiplied by the average time per instruction. Therefore, the number of clock cycles required for a program can be written as

$$\text{CPU clock cycles} = \text{Instructions for a program} \times \text{Average clock cycles per instruction}$$

We have new term - **clock cycles per instruction** (abbreviated as **CPI**), which is **the average number of clock cycles each instruction takes to execute**. Since different instructions may *take different amounts of time depending on what they do*, CPI is an average of all the instructions executed in the program. CPI provides one way of comparing two different implementations of the same instruction set architecture, since the number of instructions executed for a program will, of course, be the same. (Like use same language but different algorithm)

Hence, we can rewrite formula of basic performance equation in terms of instruction count (the number of instructions executed by the program), CPI, and clock cycle time like that

$$\text{CPU time} = \text{Instruction count} \times \text{CPI} \times \text{Clock cycle time}$$

or

$$\text{CPU time} = \frac{\text{Instruction count} \times \text{CPI}}{\text{Clock rate}}$$

These formulas are particularly useful because they separate the three key factors that affect performance. We can use these formulas to compare two different implementations or to evaluate a design alternative if we know its impact on these three parameters.

In some case, we can see different instruction types take different numbers of cycles, so above formula can rewrite

$$\text{CPU clock cycles} = \sum_{i=1}^{n} (\text{CPI}_i \times \text{Instruction Count}_i)$$

and weighted average CPI

$$\text{CPI} = \frac{\text{Clock Cycles}}{\text{Instruction Count}} = \sum_{i=1}^{n} (\text{CPI}_i \times \frac{\text{Instruction Count}_i}{\text{Instruction Count}})$$

Note:

- $\frac{\text{Instruction Count}_i}{\text{Instruction Count}}$ called **Relative frequency**
- Some book call Instruction count is C

Components of performance	Units of measure
CPU execution time for a program	Seconds for the program
Instruction count	Instructions executed for the program
Clock cycles per instruction (CPI)	Average number of clock cycles per instruction
Clock cycle time	Seconds per clock cycle

Iron Law

In Iron Law we have,

$$\text{Execution time} = \frac{\text{Time}}{\text{Program}} = \text{code size} \times \text{CPI} \times \text{cycle time} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instructions}} \times \frac{\text{Time}}{\text{Cycle}}$$

- Instructions/Program (Code size): is structions executed, not static code size and determined by algorithm, compiler, ISA (Instruction set architecture)
- Cycles/Instruction (CPI): Overlap among instructions reduces this term and determined by ISA and CPU organization
- Time/cycle (cycle time): determined by technology, organization, clever circuit design

Hardware or software component	Affects what?	How?
Algorithm	Instruction count, possibly CPI	The algorithm determines the number of source program instructions executed and hence the number of processor instructions executed. The algorithm may also affect the CPI, by favoring slower or faster instructions. For example, if the algorithm uses more divides, it will tend to have a higher CPI.
Programming language	Instruction count, CPI	The programming language certainly affects the instruction count, since statements in the language are translated to processor instructions, which determine instruction count. The language may also affect the CPI because of its features; for example, a language with heavy support for data abstraction (e.g., Java) will require indirect calls, which will use higher CPI instructions.
Compiler	Instruction count, CPI	The efficiency of the compiler affects both the instruction count and average cycles per instruction, since the compiler determines the translation of the source language instructions into computer instructions. The compiler's role can be very complex and affect the CPI in complex ways.
Instruction set architecture	Instruction count, clock rate, CPI	The instruction set architecture affects all three aspects of CPU performance, since it affects the instructions needed for a function, the cost in cycles of each instruction, and the overall clock rate of the processor.

The table illustrate the factors change IC, CPI or clock rate

Other metrics

MIPS (Million Instructions Per Second)

$$\text{MIPS} = \frac{\text{instruction count}}{\text{execution time} \times 10^6} = \frac{\text{clock rate}}{\text{CPI} \times 10^6}$$

MFLOPS (Million Floating Point Operations Per Second)

$$\text{MFLOPS} = \frac{\text{Floating point operations (FP ops) in program}}{\text{execution time} \times 10^6}$$

Assuming FP ops independent of compiler and ISA, so

- Often safe for numeric codes (matrix size determines the number of of FP ops/program)
- But sometime not always safe like missing instructions (e.g. FP divide) or optimizing compilers

Problems with MIPS

- Ignores program
- Usually used to quote peak performance with ideal conditions and **guaranteed not to exceed**

We should use MIPS when same compiler, same ISA

Amdahl's Law

> A rule stating that the performance enhancement possible with a given improvement is limited by the amount that the improved feature is used. It is a quantitative version of the law of diminishing returns.

$$\text{Speedup} = \frac{\text{old time}}{\text{new time}} = \frac{\text{new rate}}{\text{old rate}}$$

So now, let an optimization speed fraction is f of time by a factor of s

$$\text{new time} = (1 - f) \times \text{old time} + f \times \frac{\text{old time}}{\text{s}}$$

or

$$\text{speedup} = \frac{\text{old time}}{(1 - f) \times \text{old time} + f \times \frac{\text{old time}}{\text{s}}} = \frac{1}{1 - f + \frac{f}{s}}$$

Some case, s may be come to ∞, so we have

$$\lim_{x \to +\infty} \frac{1}{1 - f + \frac{f}{s}} = \frac{1}{1 - f}$$

Power

In CMOS IC technology

$$\text{Power} = \text{Capacitive load} \times \text{Voltage}^2 \times \text{Frequency}$$