MC732 - Computer Organization and Design

Pedro Henrique Limeira da Cruz

August 7, 2023

Contents

1 Computer Abstractions and Technology

1.1 Abstraction

Abstraction helps us deal with complexity, by hiding lower-lever implementation details. An example of that is the ISA (instruction set architecture), which is the abstraction from the hardware to the software, or like an API (commom on OS, web, etc).

1.2 Memory

Memory, which is a place to store data, can be divided into two different large types:

1. *Volatile memory*: Needs power to maintain stored data, like RAM.
2. *Non-volatile memory*: Persistent data storage, like magnetic disk, DVD, etc.

Although it might seem like better to always have non-volatile memory, it comes with a change in both paradigma (it may impact safety, configuration, etc) and pricing (it is cheaper to have large quantities of non-volitile memory, or at least it was that way).

1.3 Networks

In today's day and age, networking has become an integral part on the world of computing. It is resposible fo the communication between devices, resource sharing, non local access, etc.

Can be divided into three general areas:

1. Local Area Network (LAN): Ethernet
2. Wide Area Network (WAN): The internet
3. Wireless network: WiFi, Bluetooth, ...

1.4 Technology Trends

In general, since the instruction of computers, the evolution of the processing power of the CPUs have basically grown exponentialy. Th speed of the memories, however, have grown at a much slower rate (almos linear in comparision). That results in what its called *the memory wall*, which is the name given to the discrepancy between the speeds of the processors and memories.

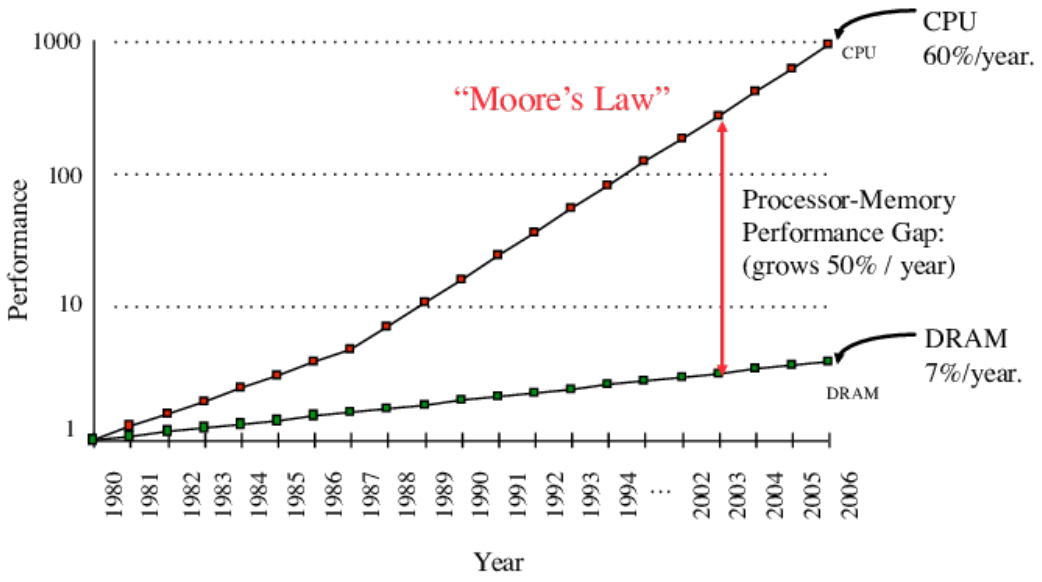

Figure 1: Performance of Memory vs Processor over the years

1.5 Performance

1.5.1 Indicators

Although when we talk about performance, we always associate it with speed, that might not give the whole picture. A clear example of that is the bellow chart, describing different types of performance metrics that might be used to characterize an airplane, which gives a general idea that there are different ways of measuring how a system behaves, and that each might be interesting to a different stakeholder.

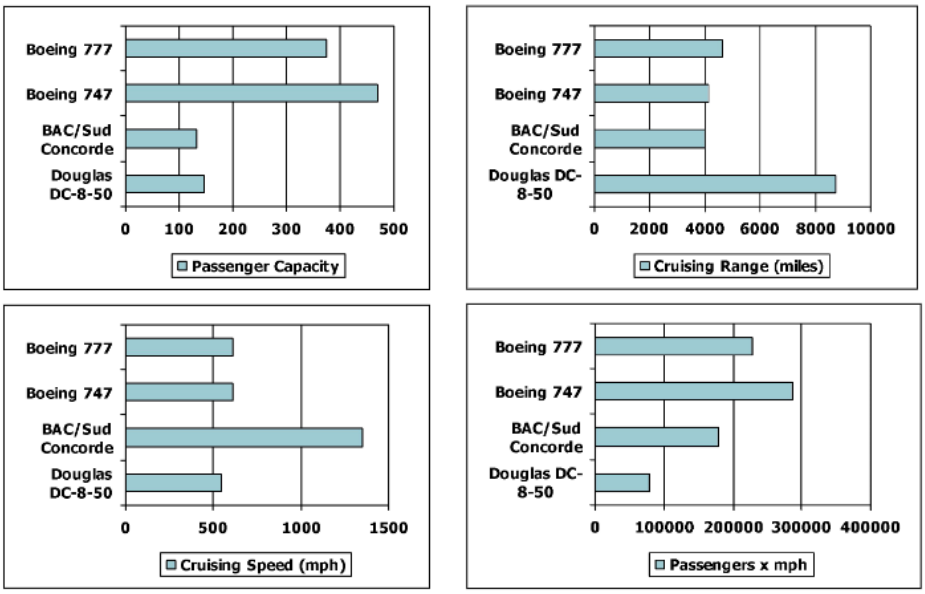

Figure 2: Different Performance Metrics for Airplanes

When it comes down to processor/computer systems, the two main indicators are:

1. Response Time: How long it takes to do a task, which might be important for the processor that runs the flight controller of an airplane.
2. Throughput: Total work done per unit time, which might be important for a server being requested hundreds of times a second.

1.5.2 Relative Performance

When choosing the right hardware/system for your problem, it is important to be able to have metrics to compare the two. Taking into account *Execution Time*, we might have:

- *Performance*: Defined as $1/Execution\ Time$
- *Speed Up/Down*: We say "X is n times faster the Y", where $n = Execution\ Time_y/Execution\ time_x$ [1]

1.5.3 Measuring Execution Time

We just saw that it is important to be able to measure relative performance, but for that we need to be able to measure the time required to run a determined application or bench mark. To do so, there are a couple of ways, each with their draw backs:

- Elapsed Time: Total response time, including all external factors, that might not be related to the task at hand (e.g I/O). Used a lot when referring to user experience.
- CPU Time: Time spent processing a given job, without considering other job's share, used heavily when analyzing scientific computing.

[1] It is important to notice that the order inverts, to say there was a *speed up*, the lowest value should be in the denominator

To check elapsed time, we can simply use an external timer, or the one on the OS. For CPU time, on the other hand, it is calculated based on the number of clock cycles and its period, shown bellow (where CLK Rate refers to the Clocks frequency, and CLK Cycles refers to the number of cycles required for the application, which some times it's refered to as *CLK Count*):

$$CPU\ Time = CPU\ CLK\ Cycles \times CLK\ Cycle\ Time \tag{1}$$

$$= \frac{CPU\ CLK\ Cycle}{CLK\ Rate} \tag{2}$$

There is, however, a **direct relationship between CLK frequency and cycle count**, where, if the frequency gets too high, the cycle count must also increase, for the time per cycle is not enough for the tasks to be finished.

It is also important to note that the execution time is strictly intertwined with the environment which the code was generated (compiler, architecture, ...).

Another way of calculating the CPU time is by using the *Cycles per instruction*, which is an average number of clock cycles needed for instructions. The CPU time is given by:

$$CPU\ Time = \frac{Instruction\ Count \times CPI}{CLK\ Rate} \tag{3}$$

If different instruction classes existis, however, the CPI becomes a weighted average:

$$CPI = \frac{CLK\ Cycles}{Instruction\ Count} = \sum_{i=1} \left(CPI_i \times \underbrace{\frac{Instruction\ Count_i}{Instruction\ Count}}_{Relative\ Frequency} \right) \tag{4}$$

CPU TIME EXAMPLE → A computer A has a 2GHz clock, and a CPU Time of 10s. The target is to develop a computer B, that aims to have a 6s CPU time and a faster clock, which results in a clock count of $1.2 \times clock\ cycles$. How fast must computer B's clock be?

- When he asks about "fast" he actually means the clock rate, which is give by:

$$CLK\ Rate_B = \frac{CLK\ Cycle_B}{CPU\ Time_B} = \frac{1.2 \times CLK\ Cycles_A}{6s}$$

- As we can see, we need to calculate the clock cycles of A, which is described by:

$$CLK\ Cycles_A = CPU\ Time_A \times CLK\ Rate_A$$
$$= 10s \times 2GHz = 20 \times 10^9$$

- Therefore we have the following clock rate for B:

$$CLK\ Rate_B = 4GHz$$