



**POLITECNICO
DI TORINO**

Recap Computer Architectures (02LSEOV)

Jacopo Nasi
Computer Engineer
Politecnico di Torino

I Period - 2017/2018

10 ottobre 2017

Indice

1	Introduction Computer Design	4
1.1	Computer Evolution	4
1.2	Designing	5

License

This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License.

You are free:

- **to Share:** to copy, distribute and transmit the work
- **to Remix:** to adapt the work

Under the following conditions:

- **Attribution:** you must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work)
- **Noncommercial:** you may not use this work for commercial purposes.
- **Share Alike:** if you alter, transform, or build upon this work, you may distribute the resulting work only under the same or similar license to this one.

More information on the Creative Commons website (<http://creativecommons.org>).



Acknowledgments

Questo breve riepilogo non ha alcuno scopo se non quello di agevolare lo studio di me stesso, se vi fosse di aiuto siete liberi di usarlo.

Le fonti su cui mi sono basato sono quelle relative al corso offerto (**Computer Architectures (02LSEOV)**) dal Politecnico di Torino durante l'anno accademico 2017/2018.

Non mi assumo nessuna responsabilità in merito ad errori o qualsiasi altra cosa. Fatene buon uso!

1 Introduction Computer Design

1.1 Computer Evolution

The first general-purpose computer was created in the late 40s. What now we can buy for 500\$ is equivalent (performance) to what could be bought for about \$1M in 85'.

During the years the performance growth was not linear, as you can see in figure 1, during the first 10 years the annual increase was around 25-30%/year, from the late 80s to the 2000 the growth is increased around 50%/year and, in the last few years it decrease to the 22%. Why this change during the increase?

The manufacturers have found a lot of physical problem related to the creation

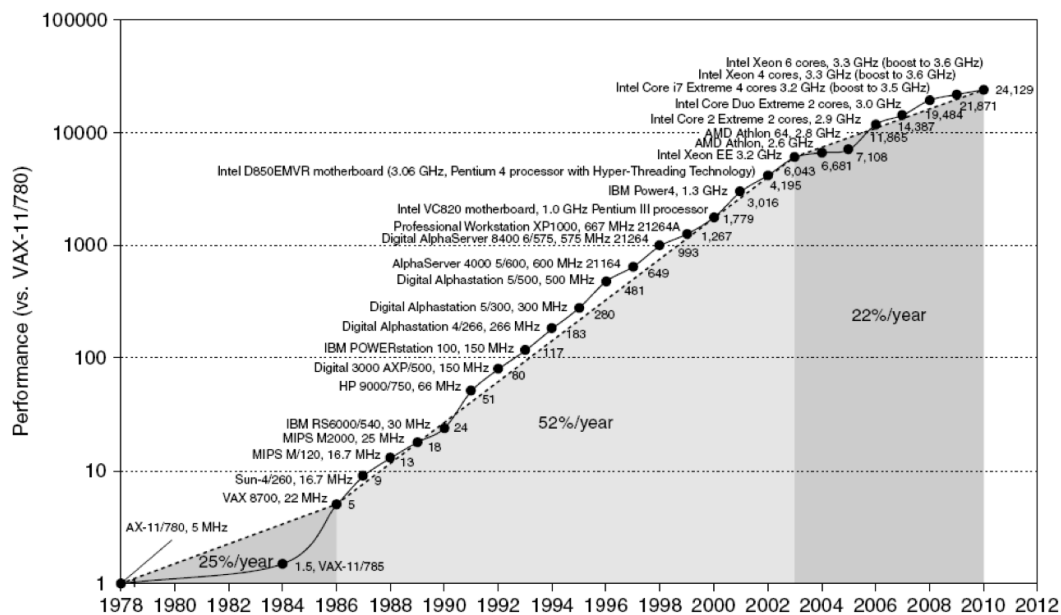


Figure 1: CPUs Growth over years

of new products, this problem are mainly related to:

- Power-Issue.
- Lower instruction-level parallelism.
- Unchanged memory latency.

in fact, since 2004, the major industry have changed the conceptual ways to desing processors, switching from single to multi-core architectures. We can say that, in anytime, this growth is incredible an is due to improvements in technology, microprocessor architecture and software development. Since the multi core introduction the major prefers to investe on multicore system rather than develop faster CPU.

1.2 Designing

There are 5 main market areas:

- **Personal Mobile Device (PMD):** Smartphone, tablet. They are focused in energy efficiency and real-time app.
- **Desktop Computing:** From PC to workstation and the main purpose is optimize the price-performance ratio.
- **Server:** Larger-scale and more reliable computing services.
- **Cluster - WAS:** Emphasis on availability, price-performance and power consumption.
- **Embedded Computers:** Fastest growing portion of PCs market. All special-purpose computer-based application, from cheap to high-end processors.

There are two **Classes of Parallelism:**

- **Data-Level (DLP):** Many data items that can be operated on at the same.
- **Task-Level (TaskLP):** Many tasks of a work can operate independently.

The first solution allows the processor to split the data operation over multiple cores, you can for example divide an array of n elements over 4 cores and, if T is the computational time needed for the entire array, the final time will be $T/4$ plus a little time for the reunion of the data. It splits the one task on different data.

The TLP instead is able to manage multiple tasks over the same data, this is the common behaviour of the actual system (pipelining techniques).

There are different Parallel Architectures:

- **Instruction-level (ILP):** This modestly uses the data-level parallelism.
- **Vector and GPU:** It exploits DLP.
- **Thread-level (TLP):** It exploits DLP and TaskLP.
- **Request-level (RLP):** Exploits parallelism among decoupled (not-related) tasks.

The designing of a new computer involves important analysis to the main purpose of it, you need to study which attributes are important for the new machine and you need to maximize performance and matching cost and power constraints. During the last decade the PC design took advantage of architectural and technology improvements, the performance increase is more than a

factor of 15 on what would have been obtained by relying solely on technology.

The **Moore's Law** says that: *The number of transistors that can be integrated into a single chip doubles every 18/24 months.* Until now the law has worked, as you can see in the figure 2 and, probably, it will work for other time.

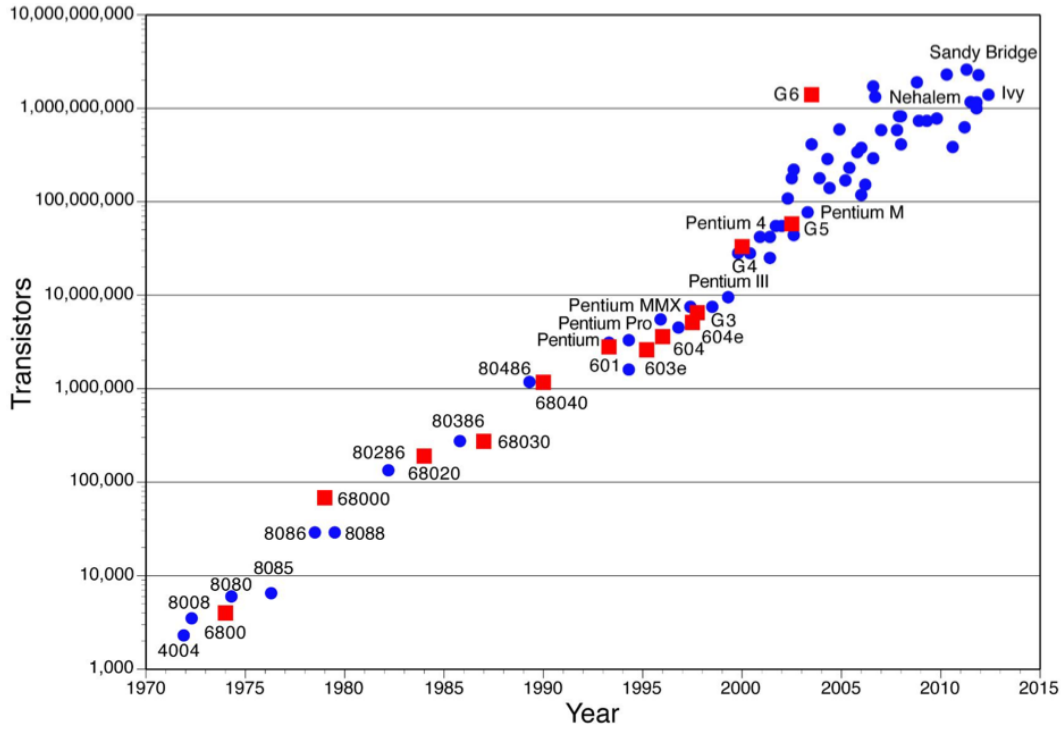


Figure 2: Transistors growth on CPUs

Cost During the evaluation of the IC manufacturing cost it is important not to forget the impact of yield, the percentage of products that pass the test phase. The production process for every product undergoes an evolution which normally leads to an improvement in yield (learning curve). The cost decrease is due to yield increase. Probably the most important part of the manufacturing cost is related to the validation and testing procedures.

Other designing problem The continuous increase of the system complexity and device integration leads to problem with power consumption, for the Power and for the Energy (mainly for portable devices). Until now the greater power consumption contribution is due to the transistor switching phase, related to the physical formulas the solution applied is trying to reduce the voltage.

Another important factor is the **Dependability** that is the quality of the

system to deliver a correct service, is traditionally very high, but it can be lowered by software or hardware bugs both during the production or in designing phase. During this years the safety-critical areas where the microprocessor have relevant importance are increased, initially only space, avionics and nuclear, but now we have rail-road, automotive, biomedical, telecom, ecc...

The dependability is often measured using:

- **Mean Time To Failure (MTTF)**: 1 failure in one billion hours.
- **Mean Time Between Failures (MTBF)**
- **Mean Time To Repair (MTTR)**

This three measures are related by the formula: $MTBF = MTTF + MTTR$

Performance they be analyzed under multiple point of view, *Time between start and completion of an operation*, *Total amount of work done in time unit*, ecc.... The UNIX system provide 4 different values:

- Elapsed time
- CPU Time
 - User
 - System

The evaluation of often performed letting work a computer and observing its behavior. Unfortunately, the choice of the application severely affects the performance and is not too easy looking the result in a correct way. The main solution is using benchmarks to compare different system running the same operation and production comparable times of execution. The benchmark sets are normally composed of:

- Kernels
- Program Fragments
- Applications

There are multiples solution to analyze the performance one of the most common is the **Amdahl's law** and it based to the comparisons with the older version of the same product. The speedup is calculated by this formula:

$$Speedup = \frac{Performance_{enhancement}}{Performance_{NOenhancement}} \quad (1)$$

the value depends on two factors:

- $Fraction_{Enhanced}$: the fraction of the computation time that take advantages of the enhancement.

- $Speedup_{Enhanced}$: the suze of the enhancement on the part it affects.

A more complete formula is:

$$Speedup_{overall} = \frac{1}{(1 - Fraction_{Enhanced}) + \frac{Fraction_{Enhanced}}{Speedup_{Enhanced}}} \quad (2)$$

an example could be useful:

An enhancement makes one machine 10 times faster for 40% of the programs the machine runs. Which is the overall speedup?

Fraction = 0.4, Speedup = 10.

$$Speedup_{overall} = \frac{1}{(1 - 0.4) + \frac{0.4}{10}} = 1.56 \quad (3)$$

Another important designing evaluation is measuring the time required to execute a program, the are severals approaches:

- Observing the real system: Not easy to be evaluated.
- Simulation: It could be really expensive.
- CPU Equation.

the latest solution use a provided formula to evaluate the CPU time:

$$CPU_{Time} = CLK_{Time} * \sum_{i=1}^n CPI_i * IC_i \quad (4)$$

where:

- CPI_i : Number of clock cycles required by instruction i (depends on hardware and instr set).
- IC_i : Number of times instruction i is executed in the program (depends instruction set and compiler).
- CLK_{Time} : Inverse of clock frequency (depends technology).

in the pipelined processor, CPI_i may vary for a given instruction, therefore the evaluation becomes much harder.