Introduction to Computer Design

M. Sonza Reorda

Politecnico di Torino
Dipartimento di Automatica e Informatica

Computer evolution

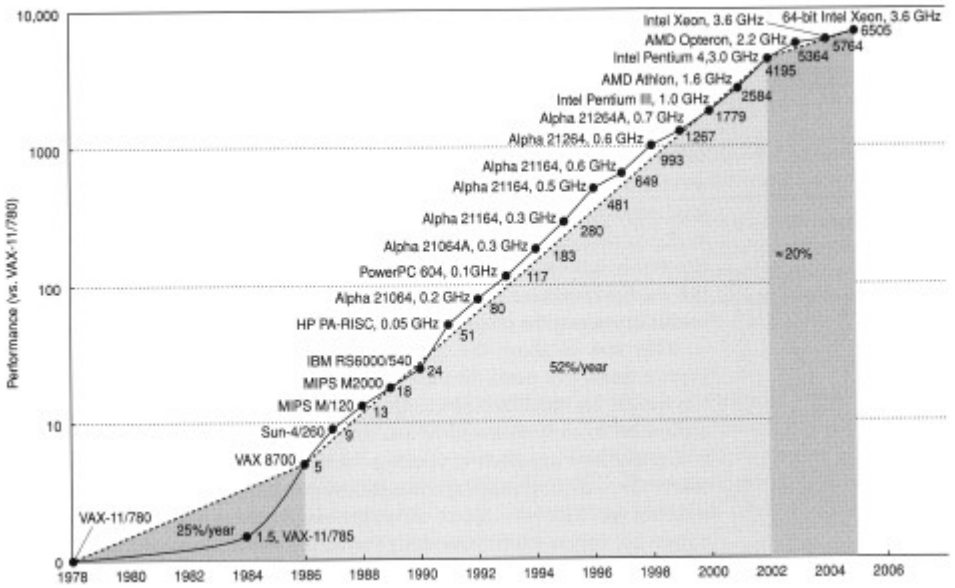
The first general-purpose computer was created in the late '40s.

A Personal Computer, that can now be bought for about 1K$, is practically equivalent (in terms of performance and memory) to what could be bought for about 1M$ in 1980.

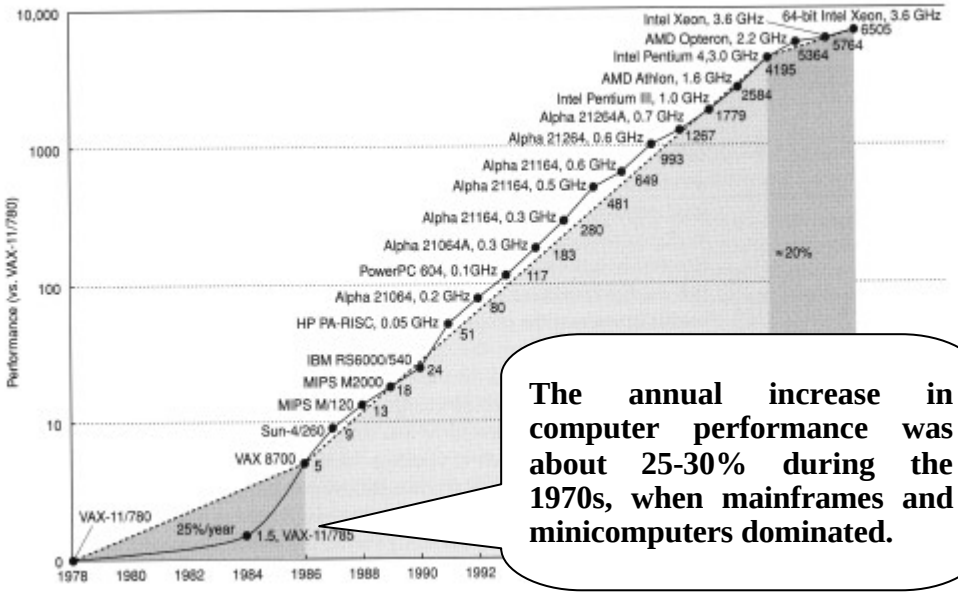
This evolution has been made possible by

- **Advances in semiconductor technology**
- **Innovations in computer design.**

Microprocessor performance



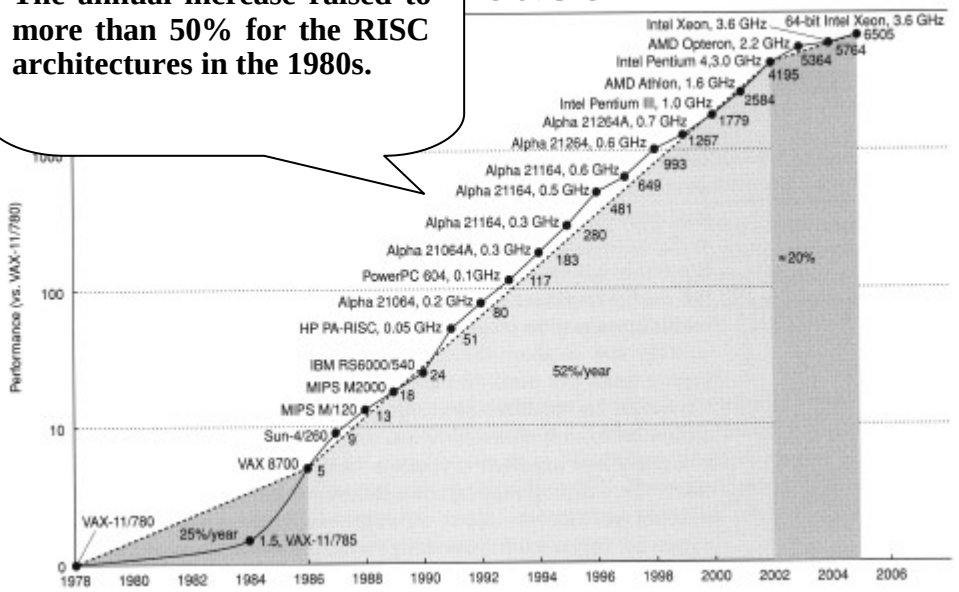
Microprocessor performance increase



The annual increase in computer performance was about 25-30% during the 1970s, when mainframes and minicomputers dominated.

Microprocessor performance increase

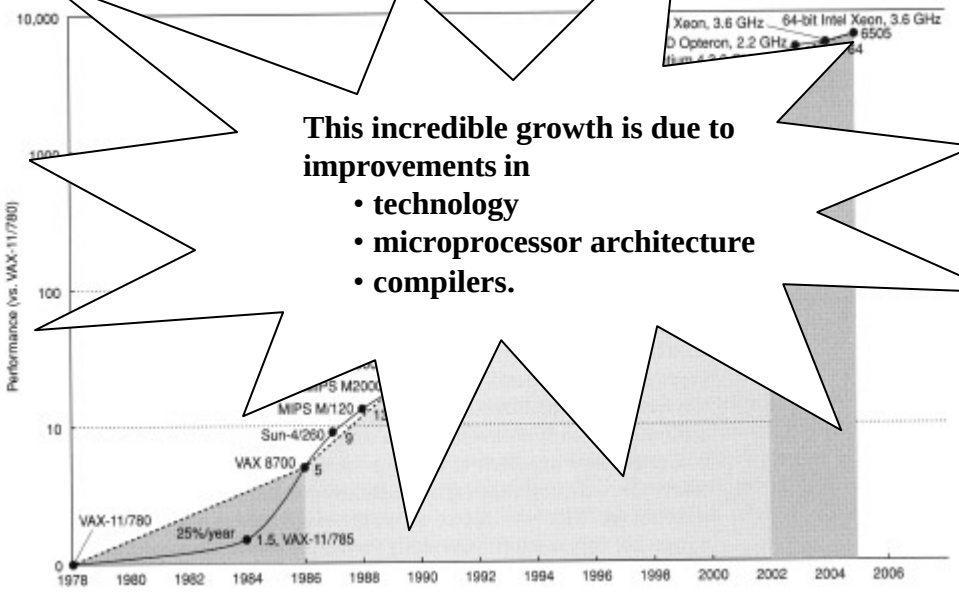
The annual increase raised to more than 50% for the RISC architectures in the 1980s.



Microprocessor performance increase

This incredible growth is due to improvements in

- **technology**
- **microprocessor architecture**
- **compilers.**



Consequences

Microprocessor-based systems are currently dominating the market of general-purpose architectures:

- ***PCs* and *workstations* dominate the market**
- ***Minicomputers* (which were used to be built using off-the-shelf logic) have been substituted by microprocessor-based servers**
- ***Main-frames* are slowly being replaced by multiprocessor systems composed of off-the-shelf processors**
- ***Supercomputers* are being built with collections of microprocessors.**

The computer market

It is currently split in 3 different areas:

- **Desktop computing**
- **Servers**
- **Embedded computers.**

Desktop computing

This area covers from PCs (~1k$) to workstations (~10k$).

The main target of this area is to optimize the *price-performance* ratio.

Servers

The main parameters of products in this area are:

- **Availability**
- **Scalability**
- **Throughput.**

Embedded computers

This area is the fastest growing portion of the computer market.

It covers all special-purpose computer-based applications (from microwaves to cell phones, from automotive to videogames).

Adopted microprocessors vary from cheap low-end 8-bit processors to very efficient (and expensive) high-end processors.

Embedded computers

Special requirements often existing in this area are

- **Real-time performance requirements**
- **Memory minimization**
- **Power consumption minimization**
- **Reliability constraints.**

Embedded computers

Embedded problems are often solved resorting to one of the following solutions:

- **Standard processor + custom logic + custom SW**
- **Standard processor + custom SW**
- **Standard DSP + custom SW.**

Programmable devices (FPGAs) are playing a growing role.

Summary

Feature	Desktop	Server	Embedded
Price of system	$1000–$10,000	$10,000–$10,000,000	$10–$100,000 (including network routers at the high end)
Price of microprocessor module	$100–$1000	$200–$2000 (per processor)	$0.20–$200 (per processor)
Microprocessors sold per year (estimates for 2000)	150,000,000	4,000,000	300,000,000 (32-bit and 64-bit processors only)
Critical system design issues	Price-performance, graphics performance	Throughput, availability, scalability	Price, power consumption, application-specific performance

Designing a computer

It means

- **determining which attributes are important for the new machine**
- **designing a machine which**
  - **maximizes performance and**
  - **matches cost constraints.**

Computer architecture

In the last decades, computer design took advantage of both

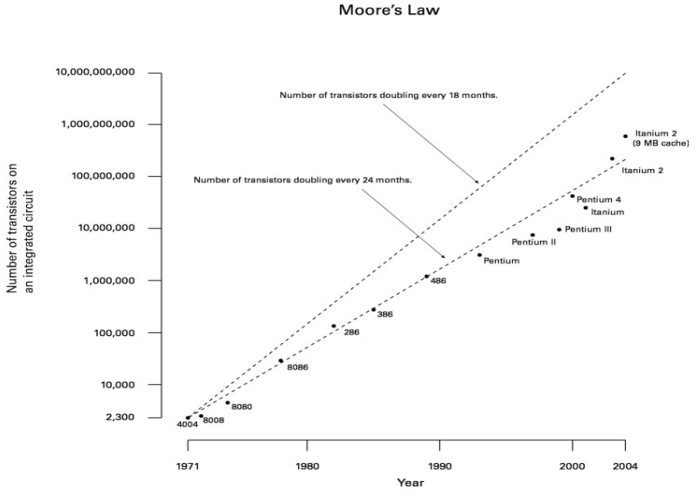
- **Architectural innovation**
- **Technology improvements.**

It was estimated that the difference between the highest-performance microprocessors available in 2001 and what would have been obtained by relying solely on technology is more than a factor of 15.

Moore's Law

The number of devices (i.e., transistors) that can be integrated into a single chip doubles every 18/24 months.

Intel processors complexity growth



Design optimization

It must take into account:

- **performance measures**
- **cost**
- **design complexity**
- **trends.**

Computer performance

What is performance?

User point of view:

performance = *response time* (time between start and completion of an operation)

System manager point of view:

performance = *throughput* (total amount of work done in the time unit).

Time

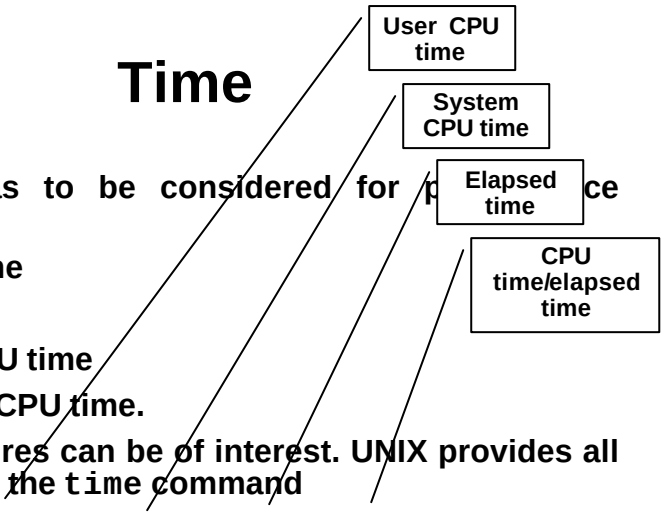
Which time has to be considered for performance computation?

- **Elapsed time**
- **CPU time**
  - **user CPU time**
  - **system CPU time.**

All these measures can be of interest. UNIX provides all of them through the `time` command

```
90.7s  12.9s  2:39  65%
```


Time



Which time has to be considered for p ce computation?

- **Elapsed time**
- **CPU time**
  - **user CPU time**
  - **system CPU time.**

All these measures can be of interest. UNIX provides all of them through the `time` command

```
90.7s  12.9s  2:39  65%
```

Performance evaluation

It is often performed by letting the computer to execute applications and observing its behavior.

Unfortunately, the choice of the applications severely affects the performance.

In the ideal case, one should use as *workload* the mix of applications the user will run.

However, they are normally unknown, and largely variable from one user to another.

Therefore, some *benchmarks* are selected to mimic real cases.

Program benchmarks

Possible benchmarks:

- ***Real programs*** **(e.g., C compilers, text processors, special-purpose tools), possibly modified**
- ***Kernels*** **(e.g., Livermore Loops, Linpack)**
- ***Toy benchmarks*** **(e.g., Quicksort, Sieve of Eratostenes)**
- ***Synthetic benchmarks*** **(e.g., Whetstone, Dhrystone).**

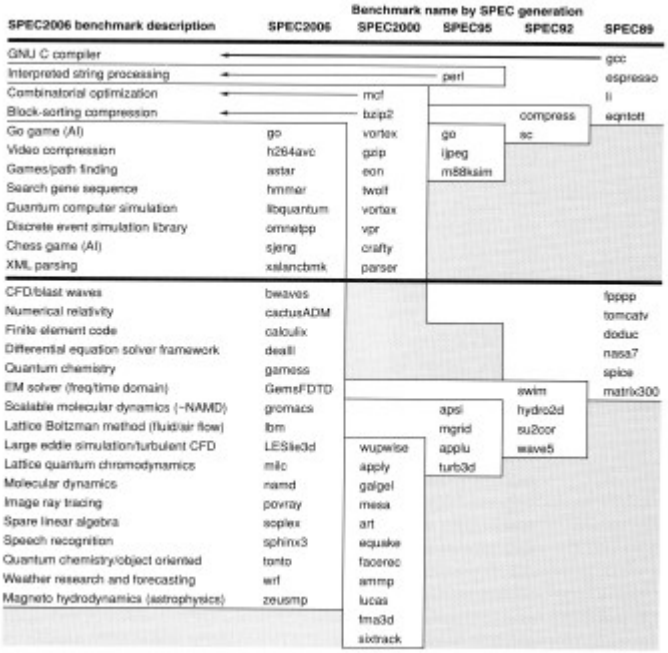
Benchmark suites

They contain a number of different programs, so that the weakness of any component is lessened by the presence of the others.

Benchmark sets are normally composed of:

- **kernels**
- **program fragments**
- **applications.**

SPEC evolution



Reproducibility

Information about execution times on benchmarks should allow reproducibility.

This means reporting detailed information about

- **hardware (system configuration)**
- **software (OS, compiler, program)**
- **program input.**

The importance of the compiler

Changing the compiler and the flags value can substantially change the performance of a program.

Therefore, the SPEC organization defined

- **a *baseline* performance: same compiler and same flags for all benchmarks**
- **an *optimized* performance: specific compiler and flags for each benchmark.**

Comparing and summarizing performance

Problem 1

I know the performance of one machine on a set of programs: which is its global performance?

Problem 2

I know the performance of two machines on the same set of programs: which is their relative performance?

A number of metrics have been proposed.

Index i runs over all benchmarks in the set

Total execution time

$$\sum_{i=1}^{n} \text{Time}_i$$

Normalized execution time

A reference machine is adopted (e.g., VAX-11/780) and execution times are normalized with respect to it.

Arithmetic and Harmonic Mean

Arithmetic mean:

$$\frac{1}{n}\sum_{i=1}^{n}\text{Time}_i$$

Harmonic mean:

$$\frac{n}{\sum_{i=1}^{n}\frac{1}{\text{Rate}_i}}$$

Weighted means

Weighted arithmetic mean:

$$\sum_{i=1}^{n}\text{Weight}_i \times \text{Time}_i$$

Weighted harmonic mean:

$$\frac{1}{\sum_{i=1}^{n}\frac{\text{Weight}_i}{\text{Rate}_i}}$$

Suggested solution

To measure a real workload and weight the programs according to their frequency of execution.

Program inputs should be carefully specified.

Guidelines and Principles for Computer Design

- **Amdahl's law**
- **CPU performance equation.**

Amdahl's Law: preliminaries

$$\text{speedup} = \frac{\text{performance with enhancement}}{\text{performance without enhancement}}$$

The speedup resulting from an enhancement depends on two factors:

- **$\text{fraction}_{\text{enhanced}}$: the fraction of the computation time that takes advantage of the enhancement**
- **$\text{speedup}_{\text{enhanced}}$: the size of the enhancement on the parts it affects.**

Amdahl's Law

$$\text{execution time}_{\text{new}} =$$

$$\text{execution time}_{\text{old}} \times ((1 - \text{fraction}_{\text{enhanced}}) + \frac{\text{fraction}_{\text{enhanced}}}{\text{speedup}_{\text{enhanced}}})$$

$$\text{speedup}_{\text{overall}} = \frac{\text{execution time}_{\text{old}}}{\text{execution time}_{\text{new}}} =$$

$$\frac{1}{(1 - \text{fraction}_{\text{enhanced}}) + \frac{\text{fraction}_{\text{enhanced}}}{\text{speedup}_{\text{enhanced}}}}$$

Example

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

$\text{fraction}_{\text{enhanced}}$ = 0.4

$\text{speedup}_{\text{enhanced}}$ = 10

$$\text{speedup}_{\text{overall}} = \frac{1}{(1-0.4)+\frac{0.4}{10}} = 1.56$$

Choosing between two solutions: an example

Two solutions are available for increasing the floating-point performance of one machine.

Solution 1

Increasing by 10 the performance of square root operations (responsible for 20% of the execution time) by adding specialized hardware.

Solution 2

Increasing by 2 the performance of all the floating-point operations (responsible for 50% of the execution time).

Which solution makes the machine faster?

Amdahl's Law application

Solution 1

$$\text{speedup}_1 = \frac{1}{(1-0.2)+\frac{0.2}{10}} = 1.22$$

Solution 2

$$\text{speedup}_2 = \frac{1}{(1-0.5)+\frac{0.5}{2}} = 1.33$$

Measuring the time required to execute a program

Possible approaches:

- **by observing the real system**
- **by simulation**
- **by applying the CPU performance equation.**

The CPU Performance Equation

$$\text{CPU time} = (\sum_{i=1}^{n} \text{CPI}_i \times \text{IC}_i) \times \text{Clock cycle time}$$

where

- ***CPI$_i$*** **is the number of clock cycles required by instruction i**
- ***IC$_i$*** **is the number of times instruction i is executed in the program**
- ***clock cycle time*** **is the inverse of clock frequency.**

The CPU Performance Equation

Depends on the hardware organization and instruction set architecture

$$\text{CPU time} = (\sum_{i=1}^{n} \text{CPI}_i \times \text{IC}_i) \times \text{Clock cycle time}$$

where

- ***CPI$_i$*** **is the number of clock cycles required by instruction i**
- ***IC$_i$*** **is the number of times instruction i is executed in the program**
- ***clock cycle time*** **is the inverse of clock frequency.**

The CPU Performance Equation

$$\text{CPU time} = \left(\sum_{i=1}^{n} \text{CPI}_i \times \text{IC}_i\right) \times \text{Clock cycle time}$$

Depends on the instruction set architecture and compiler technology

where

- ***CPI$_i$*** **is the [illegible] required by instruction i**
- ***IC$_i$*** **is the number of times instruction i is executed in the program**
- ***clock cycle time*** **is the inverse of clock frequency.**

The CPU Performance Equation

$$\text{CPU time} = \left(\sum_{i=1}^{n} \text{CPI}_i \times \text{IC}_i\right) \times \text{Clock cycle time}$$

Depends on the hardware technology and organization

where

- ***CPI$_i$*** **is the [illegible] by instruction**
- ***IC$_i$*** **is the [illegible] executed in the program**
- ***clock cycle time*** **is the inverse of clock frequency.**

Components of CPU performance

$$CPI_i = Pipeline\ CPI_i + Memory\ system\ CPI_i$$

The computation of *Pipeline CPI_i* and *Memory system CPI_i* generally requires detailed simulation of the processor and of the memory system.

Improved CPU performance equation

CPU time = (CPU clock cycles + memory stall cycles)
´ clock cycle

Memory stall cycles = IC
´ Memory references per instruction
´ Miss rate
´ Miss penalty

Example

Given the following information

- **CPI = 2.0 when all memory accesses hit in the cache**
- **load and stores account for 40% of instructions**
- **miss penalty = 25 clock cycles**
- **miss rate = 2%**

how much faster would the machine be if all instructions were cache hits?

Solution

For the machine that always hits:

CPU time$_{ideal}$ = (CPU clock cycles + memory stall cycles) ´ clock cycle

= (IC ´ CPI + 0) ´ clock cycle

= IC ´ 2.0 ´ clock cycle

For the machine with real cache:

Memory stall cycles = IC ´ Memory references per instruction ´ Miss rate ´ Miss penalty

= IC ´ (1 + 0.4) ´ 0.02 ´ 25

= IC ´ 0.7

CPU time$_{cache}$ = (IC ´ 2.0 + IC ´ 0.7) ´ clock cycle =

= 2.7 ´ IC ´ clock cycle

Solution (cont'd)

The ratio is

$$\frac{\text{CPU time}_{\text{cache}}}{\text{CPU time}_{\text{ideal}}} = \frac{2.7 \times \text{IC} \times \text{clock cycle}}{2.0 \times \text{IC} \times \text{clock cycle}}$$
$$= 1.35$$