Arquitectura de Computadores Avançada

Computer Abstractions and Technology

António Rui Borges

Summary

- *Information society*
  - *Highlights of information revolution*
- *Computer architecture vs. computer organization*
- *Structure and function*
- *Historic perspective*
- *Classes of computer systems*
- *Classes of parallelism*
- *Amdahl's law*
- *Quantitative principles of computer design*
- *Power and energy in integrated circuits*
- *Dependability*
- *Measuring performance*
- *The processor performance equation*
- *Suggested reading*

Information society - 1

The presence of computer systems in today's society is ubiquitous.

They form the basic building block of *internet* – the communications network infrastructure that connects computers worldwide, giving rise to the concept of *global village* through which people anywhere in the planet communicate with people in any other place by sending and/or receiving messages at almost real time.

They are certainly an integral part of the smart phones and tablets one owns and stand for the laptops and the desktops one uses at home or at the workplace. Through them, one can work from home, access a myriad of services such as participating in a video conference, banking and accounting, online shopping, paying bills remotely and contacting government agencies.

Industry has enormously benefited by their use. Automation of production lines has improved the efficiency of the manufacturing processes and lowered the prices of the products. By incorporating them, products themselves have become more suited to one's needs and more 'intelligent'.

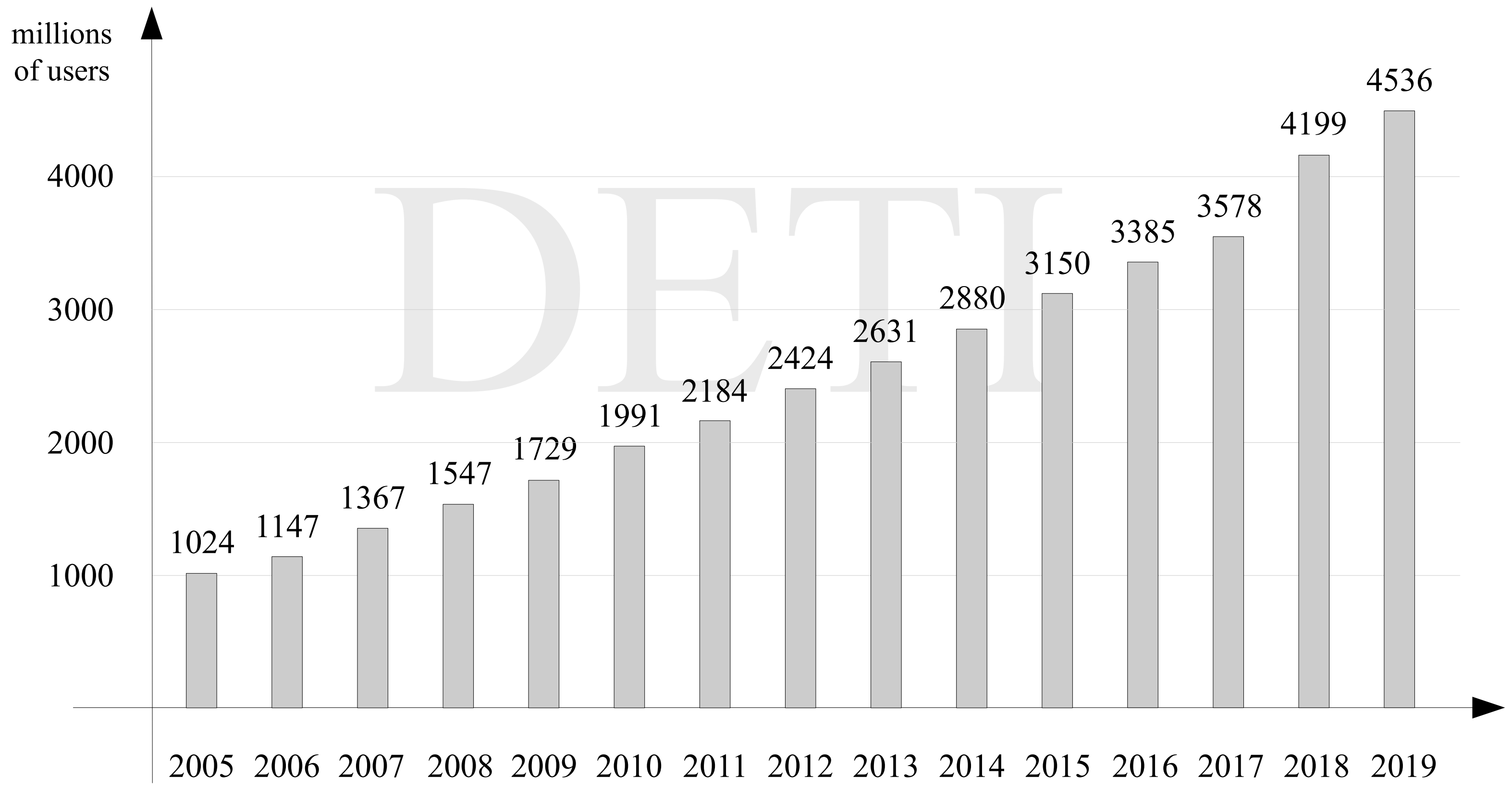
Information society - 2

Computer systems have introduced profound changes to society and the rate of change is accelerating. For instance, it is foreseen for the next few years the possibility of eliminating completely the need of physical money (coins and bank notes) and replacing it by pure electronic transactions, and of building automobiles that can drive themselves for safety reasons and comfort.

Such is their impact on day to day living that it is often said humanity is presently enduring another civilizational revolution – the *information revolution*, as compared with the *agricultural revolution* that has lead the transition from hunting and gathering to settled agriculture about 12000 years ago, and the *industrial revolution* that gave rise to new more efficient manufacturing processes in the second half of the XVIII century.

Highlights of the information revolution - 1

Internet users worldwide
Source: http://www.statista.com



Highlights of the information revolution - 2

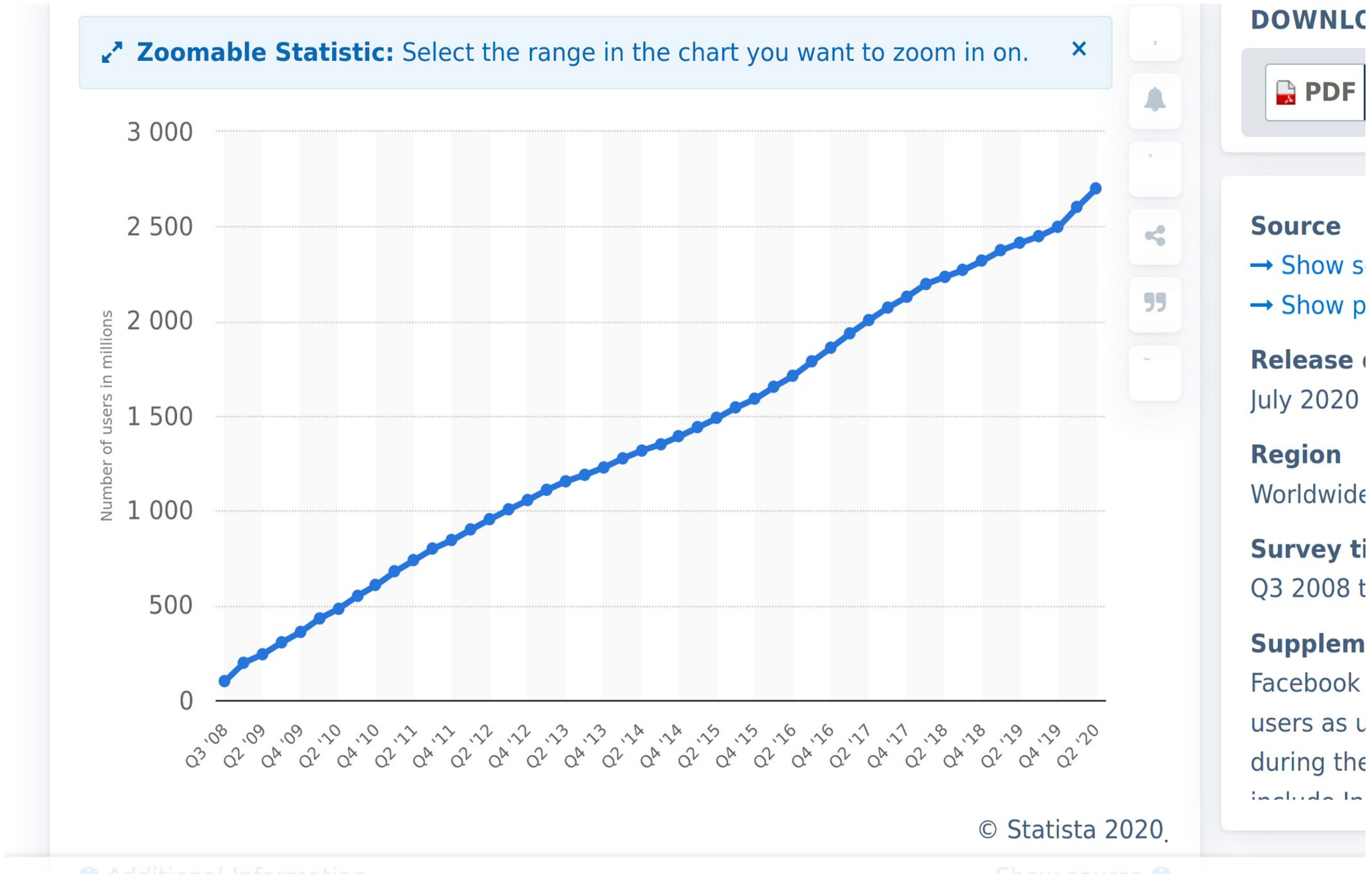
Internet users worldwide (June 2019)

Source: http://internetworldstats.com

World Regions	*Population (est.)*	*Internet Users*	*% Population*
Africa	1 320 038 716	522 809 480	39,6%
Asia	4 241 972 790	2 300 469 859	54,2%
Europe	829 173 007	727 559 682	87,7%
Portugal	**10 254 666**	**8 015 519**	**78,2%**
Middle East	258 356 867	175 502 589	67,9%
North America	366 496 802	327 568 628	89,4%
Latin America / Caribbean	658 345 826	453 702 292	68,9%
Oceania / Australia	41 839 201	28 636 278	68,4%
World Total	*7 716 223 209*	*4 536 248 808*	*58,8%*

Highlights of the information revolution - 3

Number of monthly active Facebook users worldwide
Source: http://www.statista.com

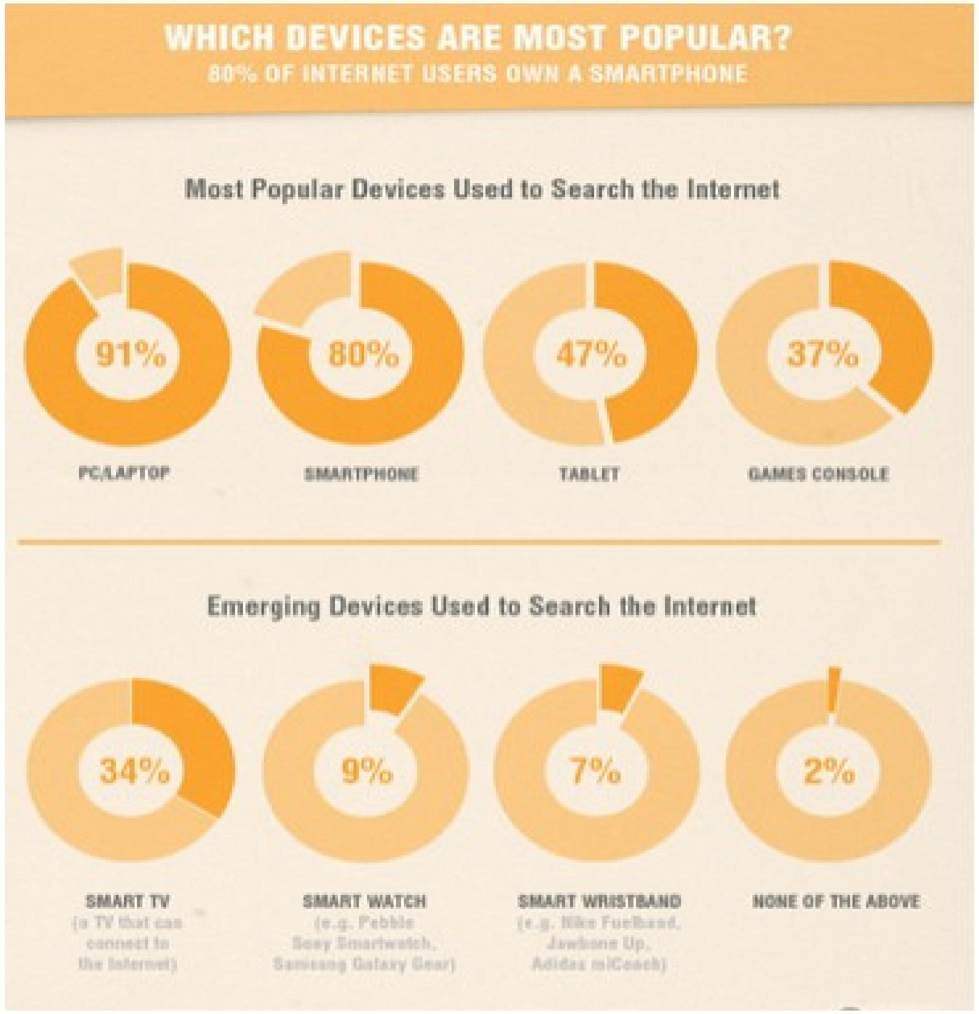


Source
➞ Show s
➞ Show p
Release
July 2020
Region
Worldwide
Survey ti
Q3 2008 t
Supplem
Facebook
users as u
during the

Highlights of the information revolution - 4

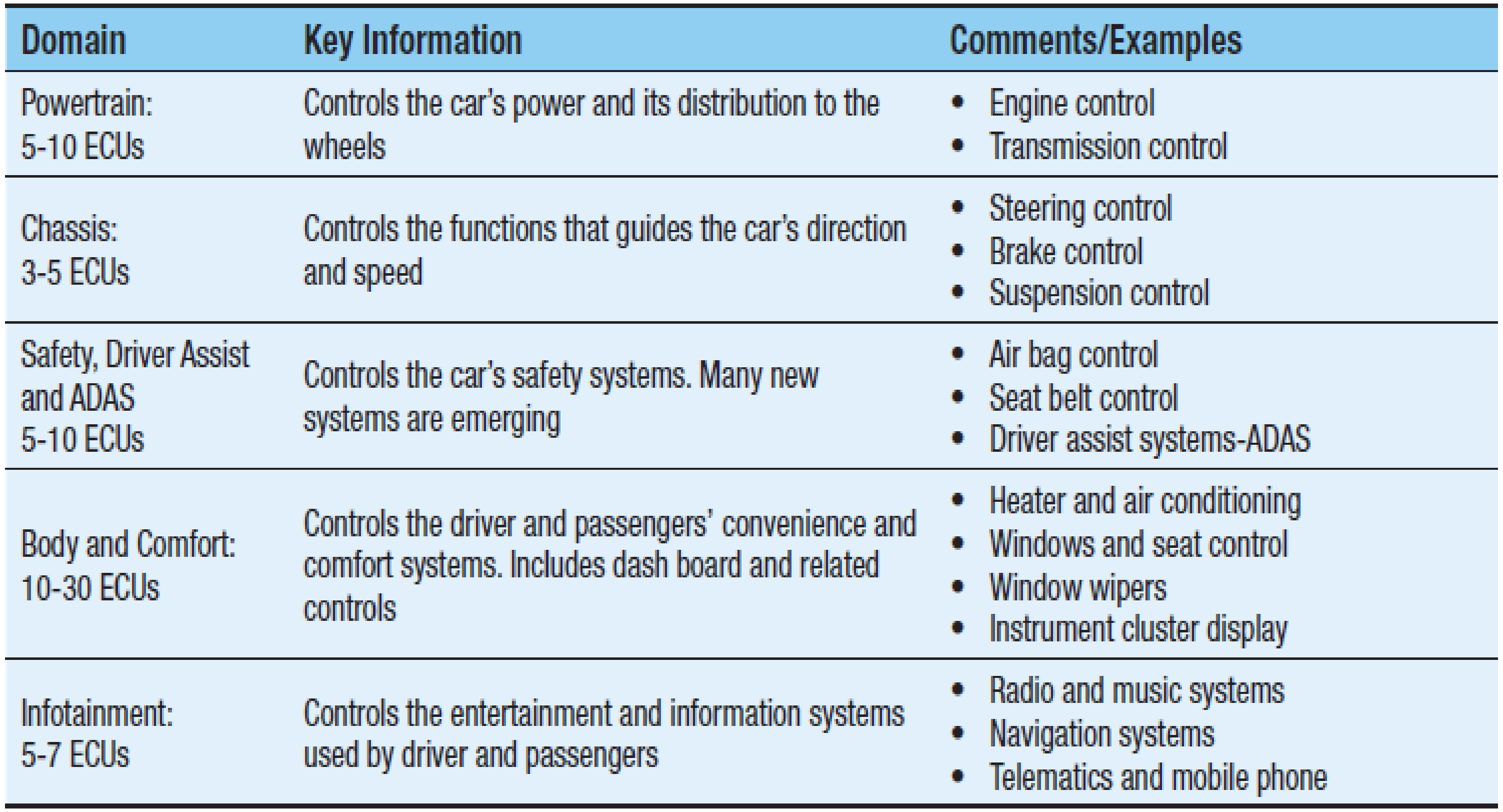
Type of access to internet (May 2016)

Source: http://www.globalwebindex.net



Highlights of the information revolution - 5

ECU Domain description

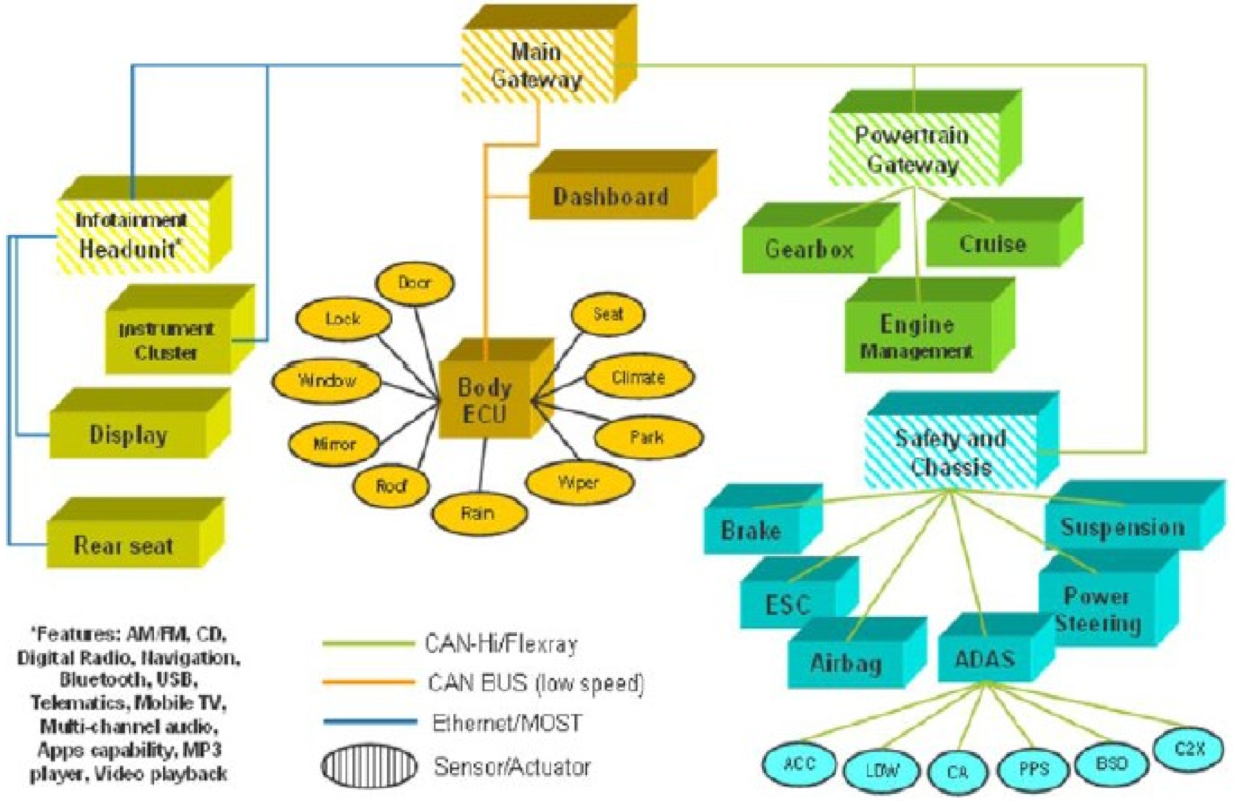


Domain	Key Information	Comments/Examples
Powertrain: 5-10 ECUs	Controls the car's power and its distribution to the wheels	• Engine control • Transmission control
Chassis: 3-5 ECUs	Controls the functions that guides the car's direction and speed	• Steering control • Brake control • Suspension control
Safety, Driver Assist and ADAS 5-10 ECUs	Controls the car's safety systems. Many new systems are emerging	• Air bag control • Seat belt control • Driver assist systems-ADAS
Body and Comfort: 10-30 ECUs	Controls the driver and passengers' convenience and comfort systems. Includes dash board and related controls	• Heater and air conditioning • Windows and seat control • Window wipers • Instrument cluster display
Infotainment: 5-7 ECUs	Controls the entertainment and information systems used by driver and passengers	• Radio and music systems • Navigation systems • Telematics and mobile phone

Source: *Is Europe in the Driver's Seat? The Competitiveness of the European Automotive Embedded Systems Industry*, Juliussen E. and Robinsonson R., Institute for Prospective Technological Studies, 2010, EUR 24601 EN

Highlights of the information revolution - 6

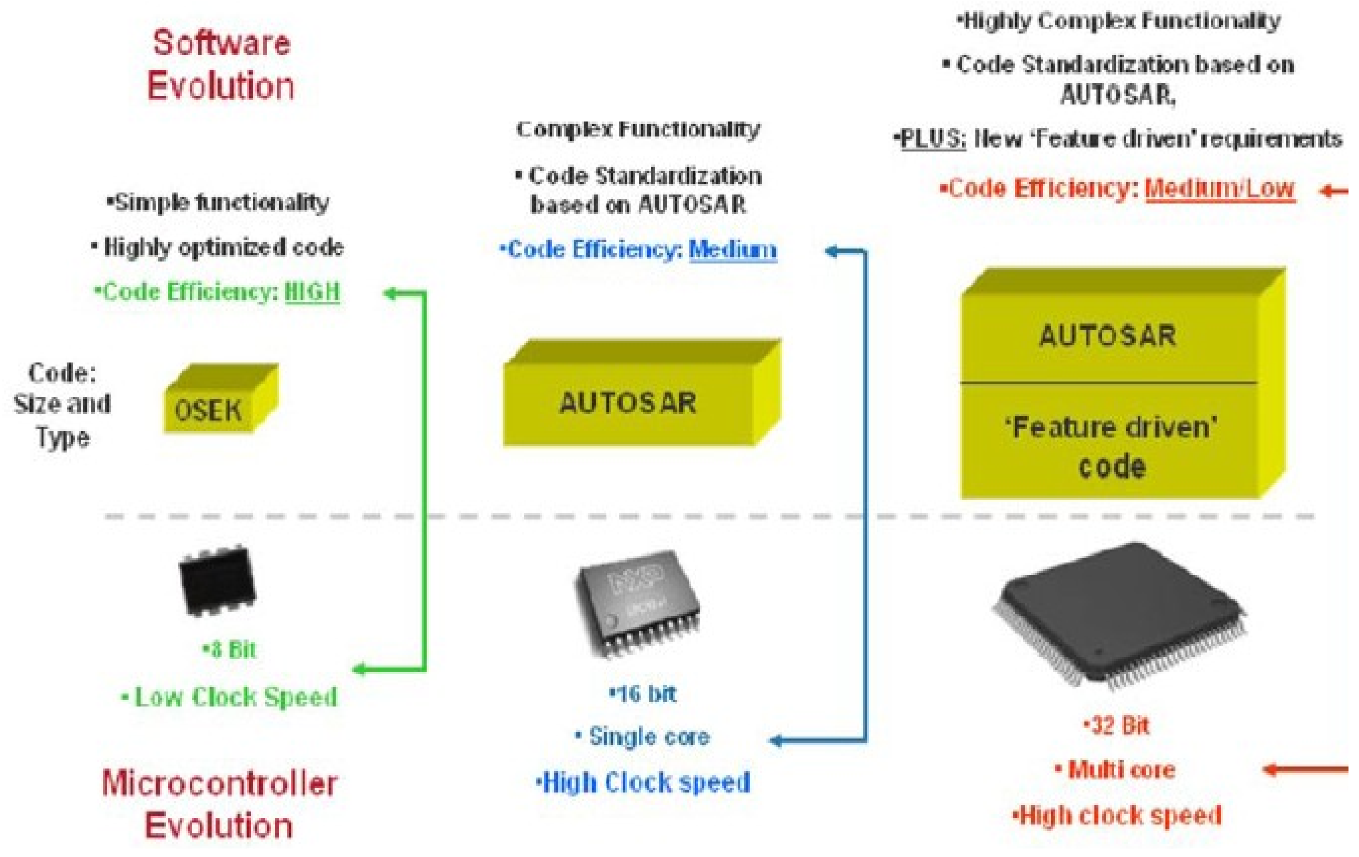
Domain controller architecture (from 2010 onwards)



Source: *Is Europe in the Driver's Seat? The Competitiveness of the European Automotive Embedded Systems Industry*, Juliussen E. and Robinsonson R., Institute for Prospective Technological Studies, 2010, EUR 24601 EN

Highlights of the information revolution - 7

Microcontroller evolution (2000 to 2020)



Source: *Is Europe in the Driver's Seat? The Competitiveness of the European Automotive Embedded Systems Industry*, Juliussen E. and Robinsonson R., Institute for Prospective Technological Studies, 2010, EUR 24601 EN

Computer architecture vs. computer organization - 1

Computer architecture – it refers to those attributes of a computer system which are visible to the programmer, that is, to those attributes which have a direct impact on the logical execution of a program.

Architectural attributes include the instruction set, the number and the size of the processor internal registers, the format of the different data types, the memory addressing modes and the I/O mechanisms.

Computer organization – it refers to the role of internal operational units and to the ways they interconnect to implement the architectural specification.

Organizational attributes include those hardware details which are transparent to the programmer such as control signals, interfaces between processor and memory and between the computer system and I/O devices (peripherals) and the memory technology used.

Computer architecture vs. computer organization - 2

The distinction between architecture and organization has been, and still is, very important. Many computer manufacturers offer a family of computer models, all with the same architecture, but with differences in organization, that is, representing different implementations of the same architecture, so that each model has different price and performance characteristics. Thus, a particular architecture may span many years and encompass a number of different computer models, its organization changing with changing technology.

A notable example in this sense is the IBM System / 370 architecture which was first introduced in 1970 and included a number of different models. The customer with modest requirements could start by buying a cheaper, slower model and, if demand increased, later upgrade to a more expensive, faster model without having to abandon the software that had already been developed. Over the years, IBM has produced many new models with improved technology to replace the older models, offering the customer greater speed, lower cost, or both. These newer models retained the same architecture, thus protecting the customer's software investment. Remarkably, the System / 370 architecture, with a few enhancements, has survived till today as the architecture of IBM's mainframes product line.

Computer architecture vs. computer organization - 3

IBM's z13 mainframe (Augusto Menezes/Feature Photo Service/IBM)

The z13 is designed to be capable of real time encryption for mobile transactions, with real time analytics processing to identify trends and help detect fraud, while the system is able to scale up to process an impressive 2.5 billion transactions per day, according to IBM. (Source: http://www.v3.co.uk)

Computer architecture vs. computer organization - 4

For microcomputers, the relationship between architecture and organization is very close. Changes in technology not only influence organization but also result in the introduction of more powerful and more complex architectures. Generally, there is less of a requirement for generation-to-generation compatibility for these smaller machines.

Structure and function - 1

A computer system is a very complex system. The key point to design and/or to describe a complex system is to use *abstraction* to express the system, or one of its parts, at different levels of representation in a hierarchical manner. At a given level, the details of lower levels are hidden so that the image that is presented is based on the concept of *what one needs to know*. By concentrating at what is relevant at each moment, fewer details have to be considered and interrelated and one becomes more productive in designing and/or gets a clearer picture and a better understanding of what is described.

At each level, one needs to be concerned with structure and function

- ***structure*** – how many components there are and the way in which they are interconnected
- ***function*** – the operation of each individual component as part of the whole.

Structure and function - 2

The basic functions of a computer system are: *inputting data*, *outputting data*, *processing data*, *storing data* and *control*.

The computer system must be able to *process data*. Data takes a wide variety of forms and the range of processing requirements is broad. There are, however, only a few fundamental methods or types of data processing.

The computer system must also be able to *transfer data* between itself and the outside world. The operating environment consists of devices that either serve as sources or destinations of data. When data are received from or delivered to a device that is directly connected to a computer system, the process of doing this is known as performing *input – output* (I/O) and the device is referred to as a *peripheral*. When data are transferred over longer distances, to or from a remote device, the process is known as *data communication*.

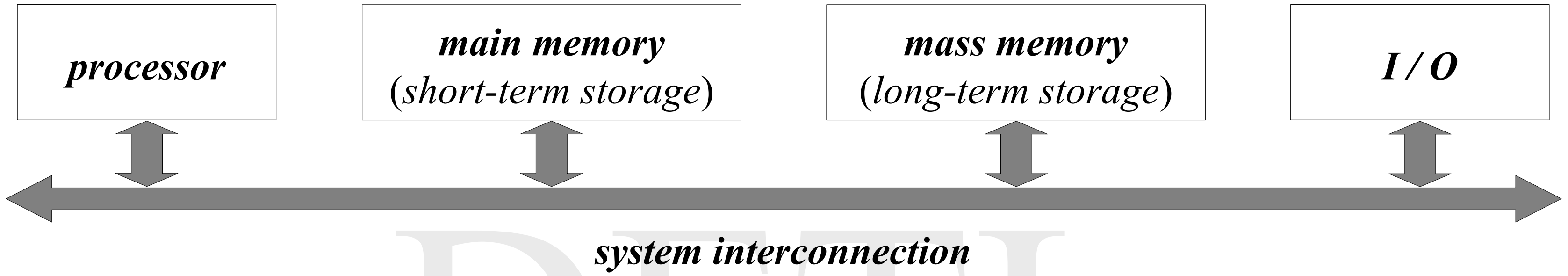
Structure and function - 3

It is essential for the computer system to *store data* too. Even if the computer is processing data on the fly (that is, data are inputted, get processed and the results are immediately outputted), the computer system must temporarily store at least those pieces of data that are being worked upon at any given moment. Thus, there is a *short-term data storage* function. Equally important, the computer system requires a *long-term data storage* function for data files to be stored and subsequently be retrieved and updated.

Finally, these functions must be *controlled* to achieve some useful work. This control is ultimately exercised by the individual (*programmer*) who provides the computer system with *instructions*, but within the computer system itself a *control unit* must manage the available resources and orchestrate the performance of its functional parts in response to those instructions.

Structure and function - 4

At top level, the structure of a computer system may be perceived as


processor
main memory (short-term storage)
mass memory (long-term storage)
I / O
system interconnection

- ***processor*** (or ***central processing unit*** – CPU) – controls the operation of the computer and performs its data processing functions
- ***main memory*** – stores data during processing; it has a *volatile* nature
- ***mass memory*** – stores data in-between processing runs and allows that large amounts of data be retrieved and eventually updated during processing; it has a *non-volatile* nature and is perceived as a special I/O device
- ***I/O*** – moves data between the computer system and its external environment
- ***system interconnection*** – ensures that data transfer takes place among the components; it is usually implemented as a *bus*.

Structure and function - 5

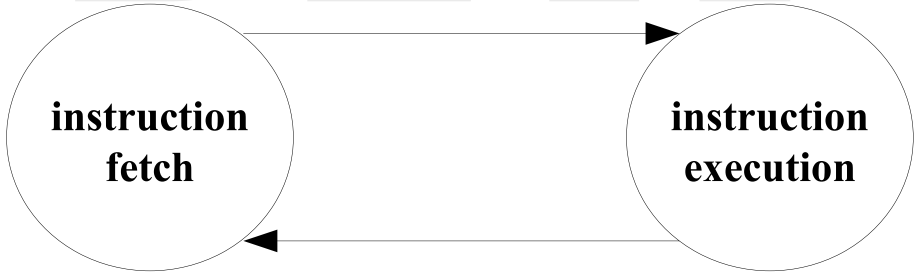
In order to have the computer system to carry out a specific task, one has to supply a group of instructions that taken together constitute the *program* to be executed. The key idea here is how to represent instructions?

Suppose they could be represented in a form suitable for being stored in the main memory alongside the data, thus constituting a *special* kind of data in some abstract sense. Then, a computer system could get its instructions by reading them from the main memory and a program could be set or altered by setting the values of that portion of the memory.

This idea, developed independently by John von Neumann and Alan Turing, has come to be known as the *stored-program concept* and has been universally adopted by computer designers. This is the reason why computer systems are sometimes called *von Neumann machines*.

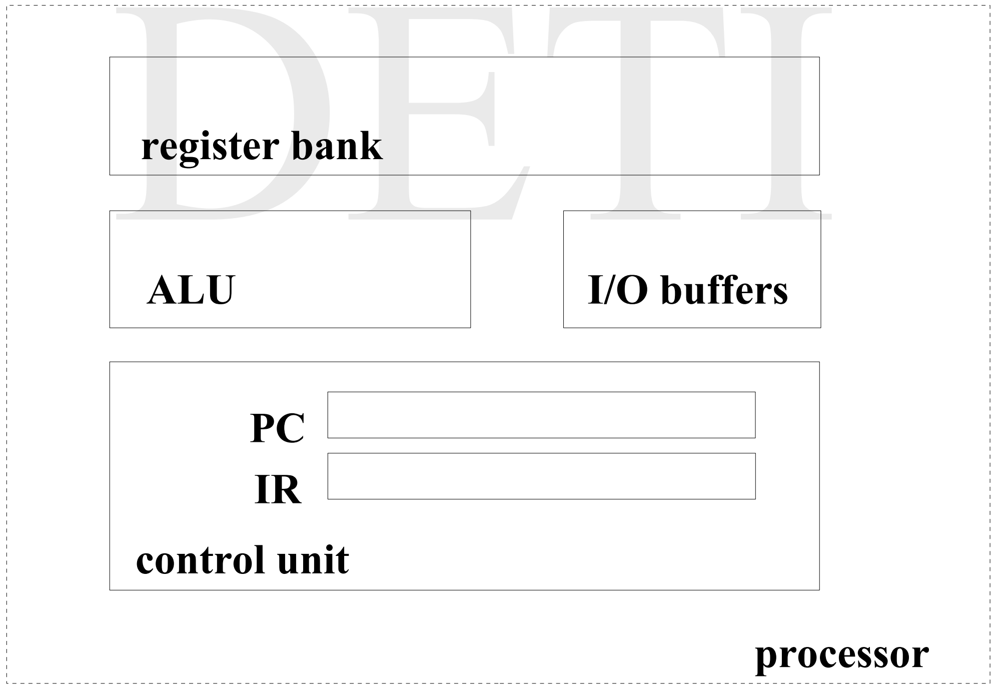
Structure and function - 6

In this sense, although a computer system is a very complex digital system, it may be perceived as a system which toggles continuously between two basic internal states: *instruction fetch*, where the processor accesses memory to get the next instruction, and *instruction execution*, where the processor decodes the just retrieved instruction and performs the corresponding operation.



Structure and function - 7

In a simplified way, the processor itself may be seen of consisting of a *control unit*, which takes mainly care of the *instruction fetch* and the *instruction decoding* phases; of a *arithmetic* / *logic unit*, which performs the prescribed operations; of a *register bank*, which stores temporary data; and of *I/O buffers*, which enable communication with the other components of the computer system.



Structure and function - 8

The *instruction set* of any processor always comprises instructions of the type

- *data movement* – transfer of data between some register of the register bank and main memory or some I/O controller
- *arithmetic* / *logic* – arithmetic instructions (add, subtract, multiply, divide), either in fixed or floating point format, logic instructions (not, and, or, x-or) and shifting / rotating register contents
- *branching* – modifying the strict sequentiality of instruction execution, either unconditionally or dependent on some condition
- *subroutine calling* – execution of subprograms (autonomous code segments within the whole group of supplied instructions), either unconditionally or dependent on some condition.

Historic perspective - 1

Computer systems performance has made an incredible progress since the time the first general purpose electronic computer was built. This rapid pace of improvement was due not only to advances in the technology of integrated circuits, but also to advances in computer design.

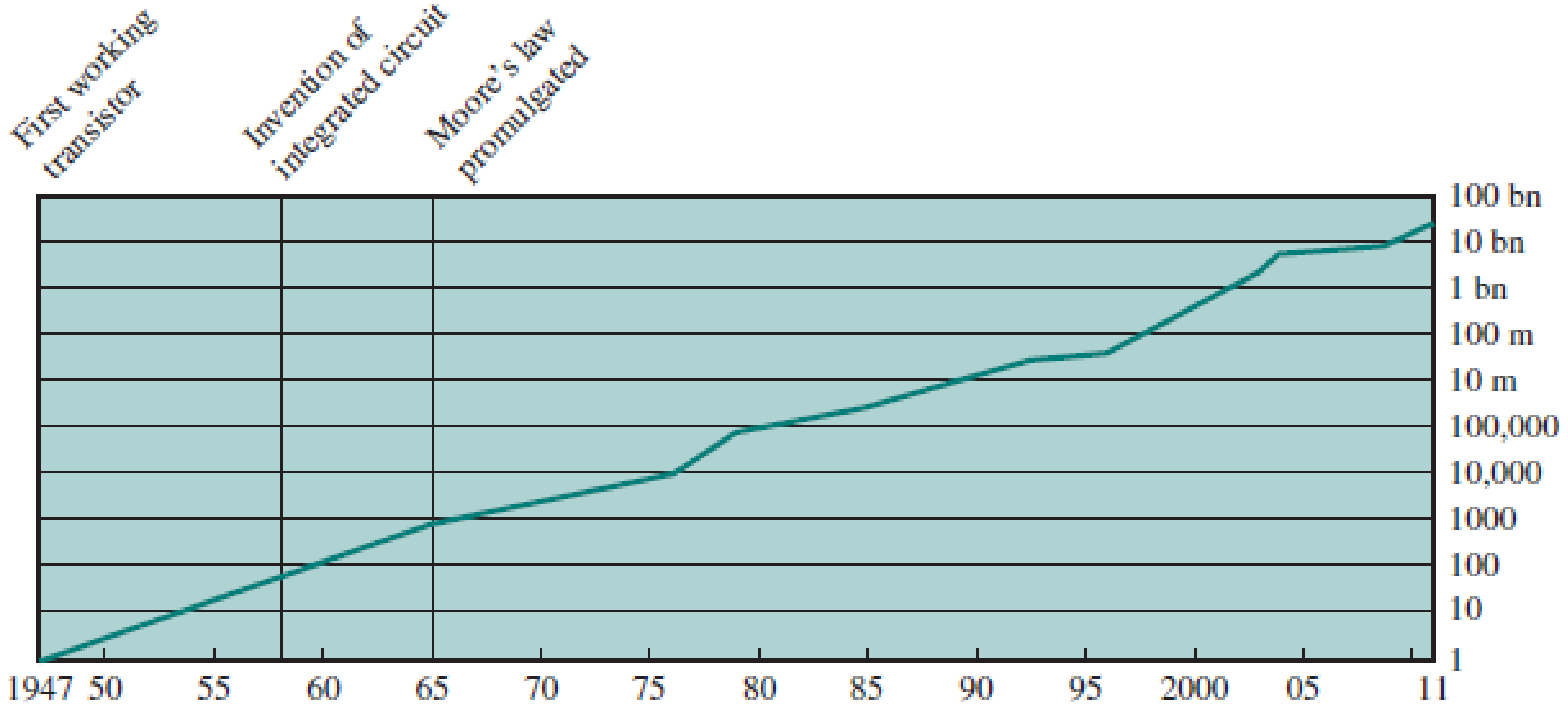
On the technological side, Gordon Moore, co-founder of Fairchild Semiconductor Corporation and Intel, observed in 1965 that the number of transistors that could be put on a single chip was doubling every year and correctly predicted that this pace would continue into the near future. This observation became known as the *Moore's Law*.

The pace went on year after year and decade after decade ever since. It has slowed somewhat to a doubling every two years in the 1970s and, more recently, 2015, Intel has announced that the pace is presently about a doubling every two and half years.

Historic perspective - 2

Growth in transistor count on integrated circuits

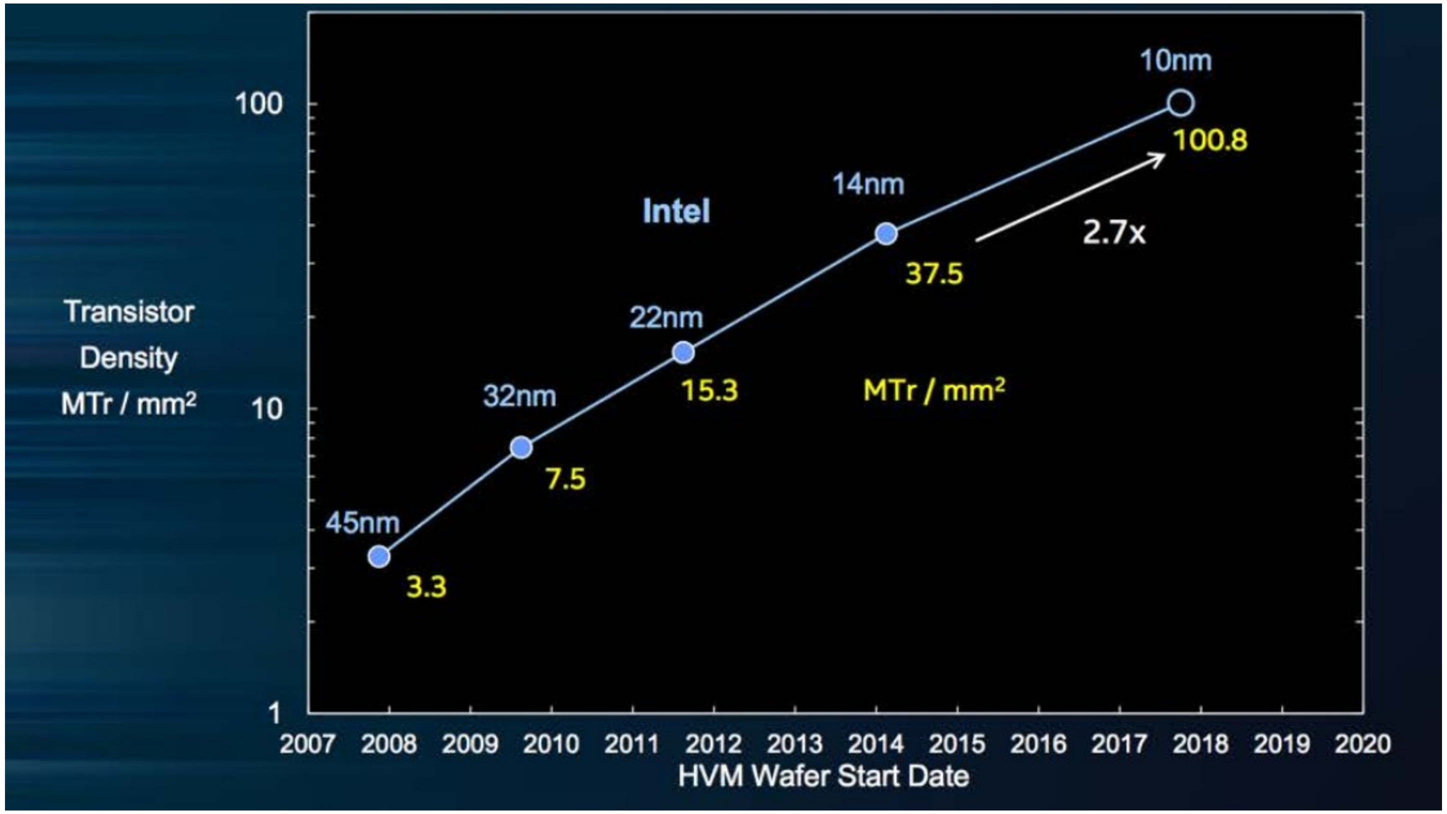
Source: Computer Organization and Architecture: Designing for Performance



Historic perspective - 3

Logic transistor density

Source: Intel's 10 nm Technology (Intel News Fact Sheet)



Historic perspective - 4

The middle 1970s saw the emergence of the microprocessor. The ability of the microprocessor to ride the improvements in integrated circuits technology and the cost advantage that resulted from a mass-produced microprocessor have led to an increasing fraction of the computer systems being based on microprocessors. In addition, two significant changes in the computer marketplace make it easier than ever before to succeed commercially with a new architecture

- the virtual elimination of assembly language programming reduced the need for object code compatibility
- the creation of standardized, vendor independent operating systems, such as Unix and later its clone Linux, lowered the cost and the risk of bringing out new processor hardware.

These changes enabled the successful development of a new set of architectures based on simpler instructions, known as RISC (Reduced Instruction Set Computers), in the early 1980s. The RISC processors focused the designers attention on two performance techniques: the exploitation of *instruction level parallelism* (first, using pipelining and, later, using multiple instruction issue) and the systematic application of caches to speed up the access of the processor to instructions and data (first, in simple forms and, later, using more sophisticated organizations and optimizations).

Historic perspective - 5

The effect of the combination of these architectural and organizational enhancements was fourfold

- it has significantly improved the computing power made available to users (the highest performant microprocessors of the day outperformed the supercomputers built 10 years previously)
- new classes of computer systems appeared due to the dramatic improvement in cost-performance: personal computers and workstations came into being in the 1980s and smart phones and tablet computers became the preferred primary computing platform of many people in the last decade
- continuing progress in semiconductor manufacturing, as predicted by Moore's Law, has led to the dominance of microprocessor-based computer systems across the entire range of computer design
- it brought forward a deep impact on software development: first, by allowing programmers to trade performance for productivity in many applications through the use of the object-oriented paradigm; second, performance is being pursued by replacing interpreters with just-in-time compilers and trace-based compiling for the traditional compiler and linker approach; third, the present popularity of Software as a Service (SaaS) over the Internet for running applications.

Historic perspective - 6

Since 2003 single-processor rate of performance improvement has decreased due to the barrier imposed by the allowed maximum amount of power dissipation of air-cooled chips and the lack of new ideas for instruction-level parallelism to be exploited efficiently.

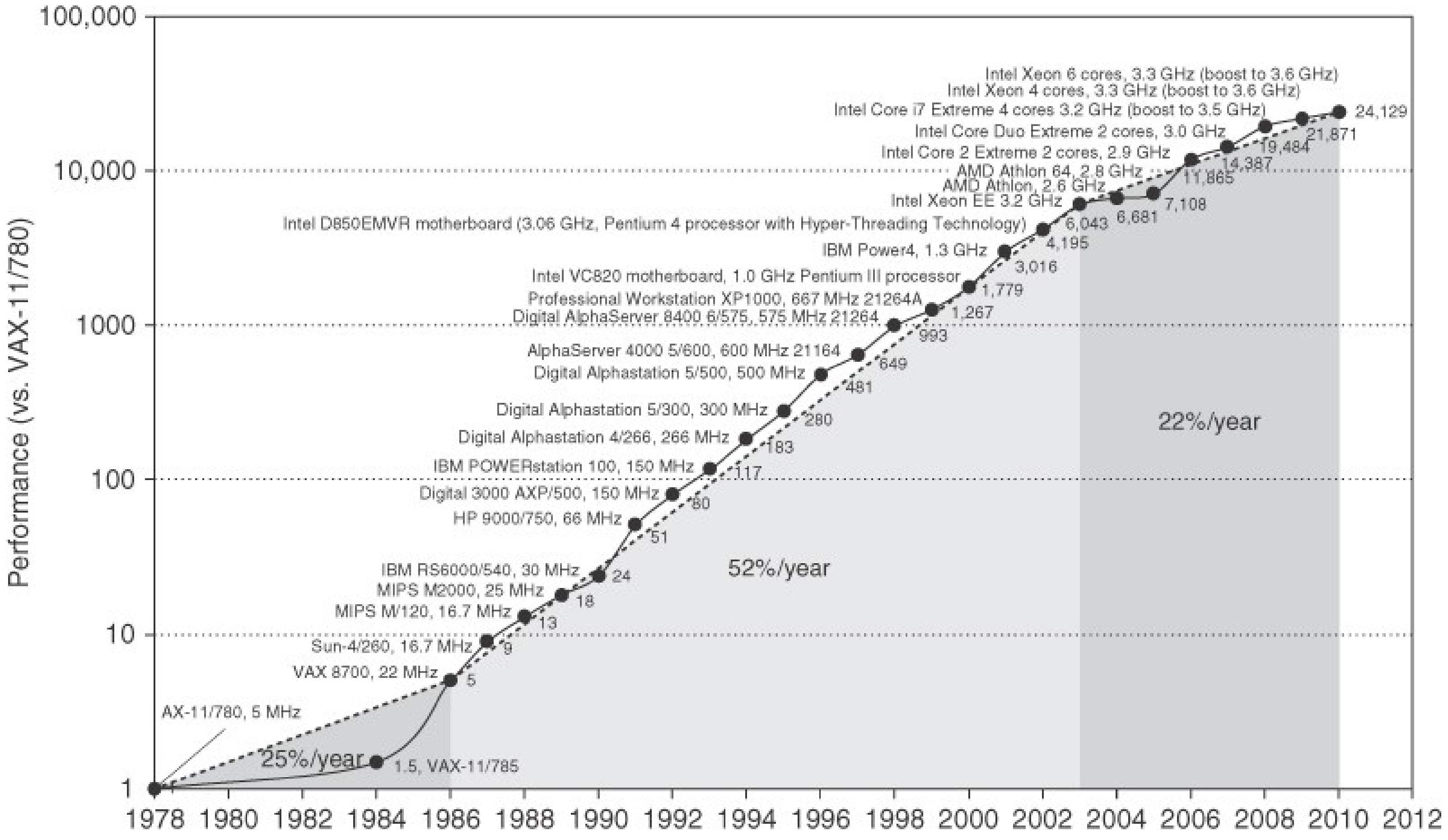
In 2004, Intel cancelled its high performance uniprocessor projects and joined other chip manufacturers in declaring that the road to higher performance would be via multicore chips rather than via faster uniprocessors.

This means that the emphasis now is placed not only in *instruction-level parallelism* (ILP), but also in *data-level parallelism* (DLP), *thread-level parallelism* (TLP) and *request-level parallelism* (RLP). Whereas hardware and compiler techniques exploit ILP implicitly without the programmer's attention, DLP, TLP and RLP are explicitly parallel and require the restructuring of applications in order to be efficiently exploited.

Historic perspective - 7

Growth in processor performance since the late 1970s

Source: Computer Architecture: A Quantitative Approach



Classes of computer systems - 1

Changes that are taking place in computer use have led to five different application markets, each characterized by specific requirements and technologies

- *personal mobile devices* (*PMD*) – they comprise a collection of wireless devices with multimedia user interfaces, such as smart phones and tablet computers; cost is a prime concern since they are primarily consumer products; although the emphasis on energy efficiency is frequently driven by the use of batteries, the need to use less expensive packaging and the absence of a fan for cooling also limit total power consumption; energy, size and weight requirements lead to the use of flash memory for mass storage; applications are often web-based and media-oriented
- *desktop computing* – they span from low-end laptops or notebooks to high-end, heavily configured workstations; the focus is on *price-performance optimization*, both in raw computing and graphics; as a result, the newest, highest performant microprocessors and cost-reduced microprocessors often appear first in desktop systems; computing tends to be well characterized in terms of applications and benchmarking, though the increasing use of web-centric, interactive applications poses new challenges

Classes of computer systems - 2

• *servers* – they represent the backbone of large-scale enterprise computing, replacing the traditional mainframe as the provider of larger-scale and more reliable file and computing services; they came into being in the 1980s when terminals were replaced by desktops as the primary interface to central services; both availability and scalability are critical, because servers are supposed to be always active and grow in response to an increasing demand for the services they support or an increase in functional requirements (thus, the ability to scale up the computing capacity, the main memory, the mass storage and the I/O bandwidth is crucial); servers are also designed for efficient throughput – responsiveness to individuals requests remains important, but overall efficiency and cost-effectiveness, as defined by the number of requests that can be handled per unit of time, are the key metrics to be taken into account

Classes of computer systems - 3

- *clusters / warehouse-scale computers – clusters* are collections of desktop computers, or servers, connected by local area networks to act as a single larger computer; each processing node runs its own [network] operating system and communicate with the others through a networking protocol; the largest clusters are called *warehouse-scale computers* (WSCs); price-performance and power are critical; availability is also critical, but contrary to servers where it is ensured by integrated computer hardware, WSCs use redundant inexpensive components as the building blocks, relying on a software layer to detect and isolate the many failures which occur; focus is placed on interactive applications, large-scale storage, dependability and high Internet bandwidth – they represent the building block of what is now called the *cloud*

  supercomputers and, in general, *high-performance computing* (HPC) are related to WSCs, but differ by emphasizing floating-point performance and by running large communication-intensive batch programs whose execution takes weeks at a time; very fast internal networks are critical here
- *embedded computers* – they are found in everyday machines, 'intelligent' instruments and consumer products; they are related to PMD computers, but contrary to them, they do not run externally developed software; price is the key factor – performance requirements do exist, but the primary goal is meeting the performance needs at a minimum price.

Classes of computer systems - 4

Classes of computer systems and their systemic characteristics

Source: Adapted from Computer Architecture: A Quantitative Approach

Feature	***PMD***	***Desktop***	***Server***	***C/WSC***	***Embedded***
System Price	\$100 - \$1 000	\$300 - \$2 500	\$5 000 - \$10 000 000	\$100 000 - \$200 000 000	\$10 - \$100 000
Microprocessor Price	\$10 - \$100	\$50 - \$500	\$200 - \$2 000	\$50 - \$250	\$0.01 - \$100
Critical Design Issues	cost, energy, performance, responsiveness	price-performance, energy, graphics-performance	throughput, availability, scalability, energy	price-performance, throughput, energy	price, energy, application-specific performance

Classes of parallelism - 1

Parallelism at various levels is the prevailing driving force of computer design across all classes of computers, with energy and cost being the primary constraints. There are basically two kinds of parallelism in applications

- *data-level parallelism* (DLP) – it arises when there are multiple data items that can be processed at the same time
- *task-level parallelism* (TLP) – it arises when the task to be carried out can be divided into subtasks that operate independently.

Computer hardware in turn can exploit these two kinds of application parallelism in four different ways

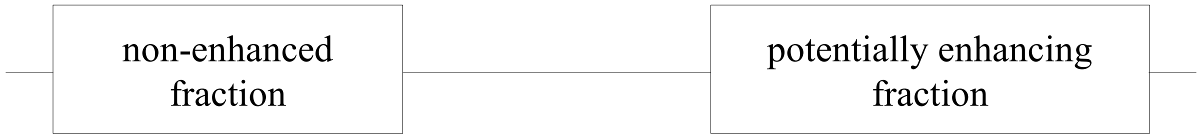
- *instruction-level parallelism* – data-level parallelism is exploited with compiler help using ideas like pipelining, speculative execution and multiple issue
- *through special hardware* – vector architectures and graphic processor units (GPUs) exploit data-level parallelism by applying the same instruction to a collection of data in parallel
- *thread-level parallelism* – data-level and task-level parallelism may be both exploited in a tightly-coupled model that allows for interaction among concurrent threads
- *request-level parallelism* – parallelism is exploited among essentially loosely-coupled tasks specified by the programmer or the operating system.

Classes of parallelism - 2

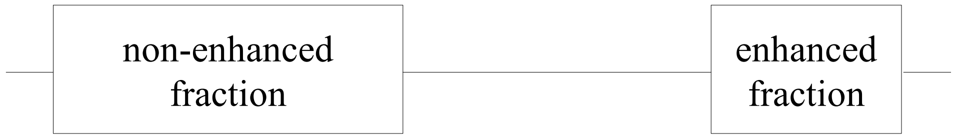
In the 1960s, Michael Flynn studied the parallel computing efforts which were made so far and found a classification that is still popular today. He looked at the parallelism in the instruction and data streams called for by the instructions at the most constrained component of the multiprocessor and placed all computers in one of the four categories

- *single instruction – single data streams* (SISD) – it corresponds to the uniprocessor; nevertheless, instruction-level parallelism can still be exploited
- *single instruction – multiple data streams* (SIMD) – the same instruction is executed by multiple processing units using different data streams; this category comprises vector architectures, multimedia extensions to standard instruction sets and GPUs
- *multiple instruction – single data streams* (MISD) – multiple instructions are executed upon the same piece of data; no commercial multiprocessor of this type has been built yet (although if the *piece of data* is considered to represent a *data vector*, then *systolic arrays* may be reasoned to fall in this category)
- *multiple instruction – multiple data streams* (MIMD) – it targets task-level parallelism where each processor runs its own program on its own data; data-level parallelism may also be exploited, although the overhead is likely to be higher than in SIMD; processor multicores fall in this category.

Amdahl's Law - 1



execution time for the entire task without using the improvement



execution time for the entire task using the improvement

Amdahl's Law - 2

The performance gain that can be obtained by improving some feature of a computer system can be estimated by the *Law of Amdahl*. Amdahl stated in 1967 that *the speed up to be gained from adopting some faster mode of execution is limited by the time fraction of the all operation where the faster mode is used* and is expressed by the formula

$$\text{speedup}_{\text{overall}} = \frac{\text{execution time for the entire task without using the improvement}}{\text{execution time for the entire task using the improvement}} =$$

$$= \frac{1}{\left(1 - \text{frac}_{\text{enhanc}}\right) + \dfrac{\text{frac}_{\text{enhanc}}}{\text{speedup}_{\text{enhanc}}}} \; ,$$

where frac_{enhanc} is the time fraction in the original computer system which can be converted to take advantage of the faster mode of execution and speedup_{enhanc} is the speed up to be gained locally by the adoption of the faster mode of execution.

Amdahl's Law - 3

The processor used in a web server is to be changed in order to speed up operations. The new processor is 10 times faster than the original one. Assuming that presently the processor is busy 40% of the time and is waiting for I/O 60% of the time, what is the overall speed up gained if the replacement takes place?

$$\text{frac}_{\text{enhanc}} = 0.4 \quad \wedge \quad \text{speedup}_{\text{enhanc}} = 10 \quad \Rightarrow$$

$$\Rightarrow \quad \text{speedup}_{\text{overall}} = \frac{1}{0.6 + \frac{0.4}{10}} = \frac{1}{0.64} = 1.56 \quad .$$

Thus, Amdahl's Law is a law of diminishing returns!

Amdahl's Law - 4

Amdahl's Law is particularly useful in comparing the overall system performance of different alternatives.

A common operation required in graphics processors is *square root*. Suppose that floating point square root (FPSQRT) is responsible for 20% of the execution time of a critical graphics benchmark. One proposal is to enhance the FPSQRT hardware and speed up 10 times this operation. The other alternative is just to make all floating point operations run faster by a factor of 1.6. Note that floating point operations account for half of the execution time of the application. Compare these two design alternatives.

$$\text{speedup}_{\text{FPSQRT}} = \frac{1}{0.8 + \frac{0.2}{10}} = \frac{1}{0.82} = 1.22$$

$$\text{speedup}_{\text{FP}} = \frac{1}{0.5 + \frac{0.5}{1.6}} = \frac{1}{0.81} = 1.23 \ .$$

Quantitative principles of computer design

Some guidelines are useful in the design of computer systems

- *take advantage of parallelism* – parallelism is one of the most important methods for improving performance; one can resort to redundancy to increase dependability and/or sometimes make operations go faster by simply adding more resources – *scalability*, or by eliciting the underlying *concurrency*; this can be done at various abstraction levels: system, individual processor or low-level digital design
- *take advantage of the principle of locality* – programs tend to reuse instructions and data that they have recently used
- *focus on the common case* – in making a design trade-off, favor the frequent case over the infrequent one; the frequent case is often simpler to optimize and can be made more rewarding; it works well not only for performance, but also for resource allocation and power.

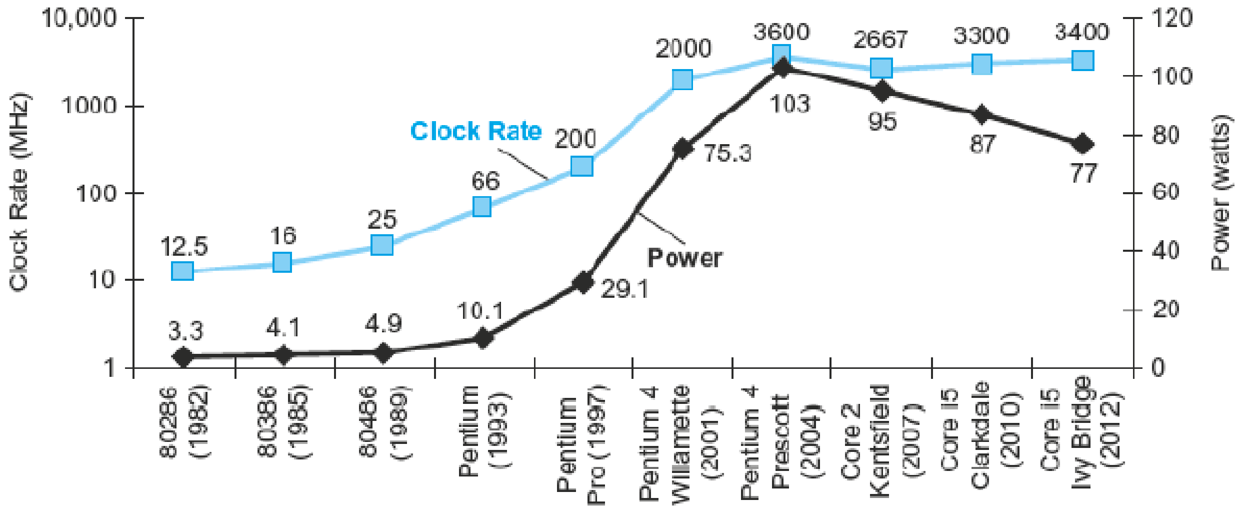
Power and energy in integrated circuits - 1

Power is the biggest challenge today facing the computer designer. First, power must be brought in and distributed around the chip and modern microprocessors use hundreds of pins and multiple interconnect layers just for power and ground. Second, power is dissipated as heat and must be removed.

Both clock rate and power increased rapidly for decades and then flattened off recently. The reason is that designers have run into the practical power limit for cooling microprocessors.

Clock rate and power for Intel x86 microprocessors over eight generations

Source: Computer Organization and Design: The Hardware / Software Interface



Power and energy in integrated circuits - 2

For CMOS chips, the traditional energy consumption has been in switching transistors, usually called *dynamic energy*. The energy required per transistor is proportional to the product of the capacitive load driven by the transistor and the square of the voltage

$$\text{energy}_{\text{dynamic}} \triangleq \text{capacitive load} \cdot \text{voltage}^2 \ .$$

The capacitive load is a function of the number of transistors connected to an output and of the technology, which by itself determines the capacitance of the wires and the transistors.

On the other hand, the *dynamic power* required per transistor is the product of the energy of a transition multiplied by the transitions frequency

$$\text{power}_{\text{dynamic}} \triangleq \text{capacitive load} \cdot \text{voltage}^2 \cdot \text{switching frequency} \ .$$

Power and energy in integrated circuits - 3

Dynamic power and energy are greatly reduced by lowering the voltage, so voltages have dropped from 5V to just under 1V in 20 years to allow for frequency increase without affecting too much the power. The problem today is that further lowering of the voltage appears to make the transistors too *leaky*.

Besides the dynamic energy consumption, one has to consider nowadays also the *static energy* consumption because of the leakage current that flows even when the transistor is turned off. In servers, for instance, the leakage current is typically responsible for 40% of the total energy consumption. Thus, increasing the number of transistors increases the need for more power dissipation even when all transistors are turned off. Server chips can consume more than 100W and cooling the chip and the surrounding system is a major expense in warehouse scale computers.

Power and energy in integrated circuits - 4

Modern microprocessors offer several techniques to improve energy efficiency despite flat clock rates and constant supply voltages

- *keep track on operations* – most microprocessors today turn off the clock of inactive modules to save energy and dynamic power
- *dynamic voltage-frequency scaling* (DVFS) – most microprocessors today offer a few clock frequencies and voltages in which to operate for lower power and energy consumption
- *design for typical case* – microprocessors to be used in desktop computing are designed for a typical case of heavy use at high operating temperatures, relying on on-chip temperature sensors to detect when activity should be reduced automatically to avoid overheating
- *over clocking* – Intel started offering *Turbo mode* in 2008, where the chip decides by itself if it is safe to run at a higher clock rate, typically 10% over the nominal clock rate, for a short period of time, possibly on just a few cores, until the temperature starts to rise.

Dependability - 1

Historically, integrated circuits were one of the most reliable components of a computer system. That conventional wisdom, however, has started to change as transistor feature sizes became smaller than 32 nm. Both transient and permanent faults are becoming more common, so computer architects must design systems to cope with these challenges.

Module reliability is a measure of continuous module operation as specified, or to put it in another way, is a measure of the time that takes a module to fail counted from a reference initial instant. Hence, the *mean time to failure* (MTTF) is a reliability measure. The reciprocal of MTTF is the rate of failures, generally reported as failures per billion hours of operation, or *failures in time* (FIT). *Service interruption* is measured by the *mean time to repair* (MTTR). *Mean time between failures* (MTBF) is just the sum of MTTF and MTTR.

If a collection of modules has exponentially distributed lifetimes – meaning that the age of a module is not relevant for its probability to fail, and their failures are independent, then the overall failure rate of the collection is the sum of the failure rates of the individual modules.

Dependability - 2

Module availability is a measure of continuous module operation as specified with respect to the alternation between service and interruption.

For nonredundant systems with repair, *module availability* is given by

$$\text{module availability} = \frac{\text{MTTF}}{\text{MTTF} + \text{MTTR}} \; .$$

Dependability - 3

Assume a disk subsystem with the following components and MTTFs

- 10 disks, each rated at 1 000 000 hour MTTF
- 1 ATA controller rated at 500 000 hour MTTF
- 1 power supply rated at 200 000 hour MTTF
- 1 fan rated at 200 000 hour MTTF
- 1 ATA cable rated at 1 000 000 hour MTTF .

Using the simplifying assumptions about the probability distributions of the life-times and the independency of failures, compute the MTTF of the entire subsystem.

$$\text{failure rate}_{\text{subsys}} = 10 \cdot \frac{1}{1000000} + \frac{1}{500000} + \frac{1}{200000} + \frac{1}{200000} + \frac{1}{1000000} =$$

$$= \frac{23}{1000000\ \text{hours}} \quad .$$

$$\text{MTTF}_{\text{subsys}} = \frac{1}{\text{failure rate}_{\text{subsys}}} = \frac{1000000}{23} \approx 43\,500 \text{ hours} \quad .$$

Dependability - 4

Disk subsystems often have redundant power supplies to improve dependability. Assume the power supply from last example, with a MTTR of 24 hours, and compute the reliability of a redundant system with two power supplies.

$$\mathrm{MTTF}_{\text{power supply pair}} = \frac{\dfrac{\mathrm{MTTF}_{\text{power supply}}}{2}}{\dfrac{\mathrm{MTTR}_{\text{power supply}}}{\mathrm{MTTF}_{\text{power supply}}}} = \frac{\mathrm{MTTF}^{2}_{\text{power supply}}}{2 \cdot \mathrm{MTTR}_{\text{power supply}}} =$$

$$= \frac{200000^{2}}{2 \cdot 24} \approx 830\,000\,000 \text{ hours} \ .$$

Dependability - 5

Amdahl's Law is also applicable beyond performance.

Consider the reliability example discussed before and assume the power supply reliability was improved from the previous 200 000 hours MTTF to the impressive 830 000 000 hours MTTF, that is, a 4 150 times improvement. How does this affect the reliability of the overall system?

$$\text{failure rate}_{\text{orig sys}} = 10 \cdot \frac{1}{1000000} + \frac{1}{500000} + \frac{1}{200000} + \frac{1}{200000} + \frac{1}{1000000} =$$

$$= \frac{23}{1000000\ \text{hours}} \;.$$

Therefore, the fraction of failure rate due to the power supply 5 per 1 000 000 hours out of 23 per 1000 000 hours, or approximately 22%. Hence, one has

$$\text{reliability improv}_{\text{power supply}} = \frac{1}{0.78 + \frac{0.22}{4150}} = 1.28 \;.$$

Measuring performance - 1

In comparing design alternatives, one often wants to relate the *performance* of two different computer systems, say X and Y. The *sentence X is faster than Y* means that the *execution time* of some program run on X is shorter than the execution time of the same program run on Y.

But the problem is how to choose a program, or a group of programs, commonly called a *benchmark*, or a *benchmark suite*, that will produce significant results in general and that do not rely on specific features of any of the alternatives.

Furthermore, one has to consider that the applications in the real world run in computer systems are so different in size and complexity that the choice of a computer system for a specific area is not an easy task.

Measuring performance - 2

One way to proceed is to run programs that are simpler than real applications, but that at the same time may be considered representative

- *kernels* – small, key portions of real applications
- *toy programs* – small programs, usually with a length up to 100 code lines, of the type students are asked to write in programming classes
- *synthetic code* – fake programs which were written with the intent to try and match the behavior of real applications.

This approach is not popular anymore, mostly because it is possible to write compilers that take the knowledge of specific programs into account so that a specific computer system appear to run these programs faster than when it is actually running real applications.

The favored approach nowadays is to consider sets of programs whose representativeness is generally accepted by a wide audience and use them to characterize the relative performance of two computer systems, one of them being the reference computer.

Measuring performance - 3

The *Standard Performance Evaluation Corporation* (SPEC) is a non-profit corporation formed to establish, maintain and endorse a standardized set of relevant benchmarks that can be applied to the newest generation of high-performance computers. SPEC develops benchmark suites and also reviews and publishes submitted results from member organizations and other benchmark licensees.

SPECfp2000 (Sun Ultra 5 as the reference computer)

Source: Computer Architecture: A Quantitative Approach

Benchmark	Ultra 5 exec time (s)	Opteron exec time (s)	SPECRatio	Itanium 2 exec time (s)	SPECRatio	Opteron/Itanium 22 exec time (s)	Itanium 2 / Opteron SpecRatio
wupwise	1 600	51.5	31.06	56.1	28.53	0.92	0.92
swim	3 100	125.0	24.73	70.7	43.85	1.77	1.77
mgrid	1 800	98.0	18.37	65.8	27.36	1.49	1.49
applu	2 100	94.0	22.34	50.9	41.25	1.85	1.85
mesa	1 400	64.6	21.69	108.0	12.99	0.60	0.60
galgel	2 900	86.4	33.57	40.0	72.47	2.16	2.16
art	2 600	92.4	28.13	21.0	123.67	4.40	4.40
equake	1 300	72.6	17.92	36.3	35.78	2.00	2.00
facerec	1 900	73.6	25.80	86.9	21.86	0.85	0.85
ammp	2 200	136.0	16.14	132.0	16.63	1.03	1.03
lucas	2 000	88.8	22.52	107.0	18.76	0.83	0.83
fma3d	2 100	120.0	17.48	131.0	16.09	0.92	0.92
sixtrack	1 100	123.0	8.95	68.8	15.99	1.79	1.79
apsi	2 600	150.0	17.36	231.0	11.27	0.65	0.65
Geometric mean			**20.86**		**27.12**	**1.30**	**1.30**

Measuring performance - 4

$$\frac{\text{Geometric mean}_A}{\text{Geometric mean}_B} = \frac{\sqrt[n]{\prod_{i=1}^{n} \text{SPECRatio } A_i}}{\sqrt[n]{\prod_{i=1}^{n} \text{SPECRatio } B_i}} = \sqrt[n]{\prod_{i=1}^{n} \frac{\text{SPECRatio } A_i}{\text{SPECRatio } B_i}} =$$

$$= \sqrt[n]{\prod_{i=1}^{n} \frac{\dfrac{\text{exec time Ref}_i}{\text{exec time } A_i}}{\dfrac{\text{exec time Ref}_i}{\text{exec time } B_i}}} = \sqrt[n]{\prod_{i=1}^{n} \frac{\text{exec time } B_i}{\text{exec time } A_i}} =$$

$$= \sqrt[n]{\prod_{i=1}^{n} \frac{\text{Performance } A_i}{\text{Performance } B_i}}$$

The processor performance equation - 1

CPU execution time for running a program can be expressed by

$$\text{CPU execution time} \; = \; \text{CPU clock cycles} \cdot \text{clock cycle time} \;\; ,$$

where the variable *CPU clock cycles* represent the total number of clock cycles for the execution of the program and the variable *clock cycle time* is period of the CPU clock.

This expression can be further expanded into

$$\text{CPU execution time} \; = \; \text{instruction count} \cdot \text{CPI} \cdot \text{clock cycle time} \;\; ,$$

where the variable *instruction count* represent the total number of instructions executed by the program and it can be obtained in a straightforward manner for non-sophisticated processor organizations if one has a listing of the program compilation into assembly language, and the variable *CPI* is simply the average number of clock cycles per instruction.

The processor performance equation - 2

CPU execution time for running a program is, therefore, made dependent of three features: instruction count for the execution of the program, average number of clock cycles per instruction and clock cycle time. Furthermore, *CPU execution time* is *equally* dependent on these three parameters: a given percent change in one of them will result in the same change for *CPU execution time*.

Unfortunately, it is difficult to change one parameter in complete isolation of the others as the basic technologies involved in the changing are interdependent

- *instruction count* – is dependent on the computer architecture and the compiler technology
- *clock cycles per instruction* – is dependent on both the computer architecture and the computer organization
- *clock cycle time* – is dependent on the underlying hardware technology and on the computer organization.

The processor performance equation - 3

CPU execution time for running a program can still be further refined if one takes into account the *CPI* for each type of instruction that is being executed

$$\text{CPU execution time} = \sum_i \left(\frac{\text{instruction count}_i}{\text{instruction count}} \cdot \text{CPI}_i\right) \cdot \text{clock cycle time} \; .$$

The processor performance equation - 4

Suppose the following measurements were made on running a benchmark in a given computer system

- frequency of FP operations = 25%
- average CPI for FP operations = 4.00
- average CPI for all other operations = 1.33
- frequency of FPSQR = 2%
- CPI of FPSQR = 20 .

Assume that design alternatives are being considered either to decrease the CPI for FPSQR to 2, or to decrease the average CPI for all other operations to 2.5. Use the processor performance equation to compare these alternatives.

Since CPI is the only parameter that changes in the processor performance equation, the answer may be given by just computing the overall CPI in the original case and for the two alternatives.

The processor performance equation - 5

Original situation

$$\text{overall CPI}_{\text{orig}} = \sum_i \frac{\text{instruction count}_i}{\text{instruction count}} \cdot \text{CPI}_i =$$

$$= 4.00 \cdot 0.25 + 1.33 \cdot 0.75 = 2.00$$

Enhanced FPSQR

$$\text{overall CPI}_{\text{enFPSQR}} = \text{overall CPI}_{\text{orig}} - 0.02 \cdot \left(\text{CPI}_{\text{origFPSQR}} - \text{CPI}_{\text{enFPSQR}}\right) =$$

$$= 2.00 - 0.02 \cdot \left(20 - 2\right) = 1.64$$

Enhanced FP

$$\text{overall CPI}_{\text{enFP}} = \sum_i \frac{\text{instruction count}_i}{\text{instruction count}} \cdot \text{CPI}_i =$$

$$= 2.50 \cdot 0.25 + 1.33 \cdot 0.75 = 1.63$$

Suggested reading

- *Computer Architecture: A Quantitative Approach*, Hennessy J.L., Patterson D.A., 5th Edition, Morgan Kaufmann, 2012
  - Chapter 1: *Fundamentals of Quantitative Design and Analysis*
  - Appendix L: *Historical Perspectives and References*
- *Computer Organization and Architecture: Designing for Performance*, Stalling W., 9th Edition, Prentice Hall, 2013
  - Chapter 1: *Introduction*
  - Chapter 2: *Computer Evolution and Performance*