Subject Name: Advanced Computer Architecture

Unit-1

1. Computer architecture can be defined as a set of rules and methods that describe the functionality, management and implementation of computers. To be precise, it is nothing but rules by which a system performs and operates.

Sub-divisions

Computer Architecture can be divided into mainly three categories, which are as follows −

- **Instruction set Architecture or ISA** − Whenever an instruction is given to processor, its role is to read and act accordingly. It allocates memory to instructions and also acts upon memory address mode (Direct Addressing mode or Indirect Addressing mode).
- **Micro Architecture** − It describes how a particular processor will handle and implement instructions from ISA.
- **System design** − It includes the other entire hardware component within the system such as virtualization, multiprocessing.

Role of computer Architecture

- The main role of Computer Architecture is to balance the performance, efficiency, cost and reliability of a computer system.
- For Example − Instruction set architecture (ISA) acts as a bridge between computer's software and hardware. It works as a programmer's view of a machine.
- Computers can only understand binary language (i.e., 0, 1) and users understand high level language (i.e., if else, while, conditions, etc). So to communicate between user and computer, Instruction set Architecture plays a major role here, translating high level language to binary language.

Structure

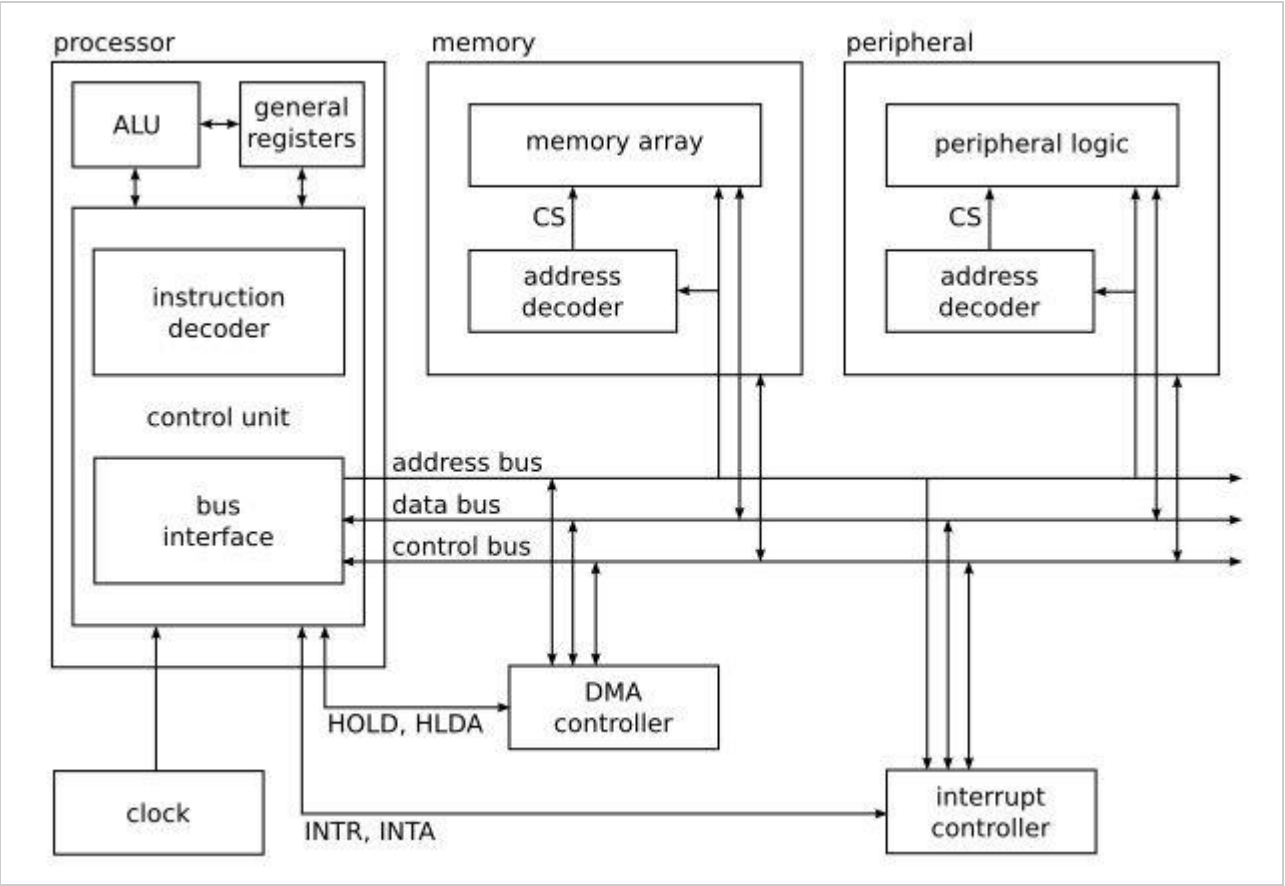
Let us see the example structure of Computer Architecture as given below.

Generally, computer architecture consists of the following −

- Processor
- Memory
- Peripherals

All the above parts are connected with the help of system bus, which consists of address bus, data bus and control bus.

The diagram given below depicts the computer architecture −



2. Technology Trends

Computer technology has made incredible progress in the roughly 70 years since the first general-purpose electronic computer was created. Today, less than $500 will purchase a cell phone that has as much performance as the world's fastest computer bought in 1993 for $50 million. This rapid improvement has come both from advances in the technology used to build computers and from innovations in computer design.

- During the first 25 years of electronic computers, both forces made a major contribution, delivering performance improvement of about 25% per year.
- The late 1970s saw the emergence of the microprocessor. The ability of the microprocessor to ride the improvements in integrated circuit technology led to a higher rate of performance improvement—roughly 35% growth per year.
- This growth rate, combined with the cost advantages of a mass-produced microprocessor, led to an increasing fraction of the computer business being based on microprocessors.

- Two significant changes in the computer market place made it easier than ever before to succeed commercially with a new architecture.

First, the virtual elimination of assembly language programming reduced the need for object-code compatibility.

Second, the creation of standardized, vendor-independent operating systems, such as UNIX and its clone, Linux, lowered the cost and risk of bringing out a new architecture.

- These changes made it possible to develop successfully a new set of architectures with simpler instructions, called RISC (Reduced Instruction Set Computer) architectures, in the early 1980s.
- The RISC-based machines focused the attention of designers on two critical performance techniques, the exploitation of instruction-level parallelism (initially through pipelining and later through multiple instruction issue) and the use of caches (initially in simple forms and later using more sophisticated organizations and optimizations).
- The RISC-based computers raised the performance bar, forcing prior architectures to keep up or disappear.
- As transistor counts soared in the late 1990s, the hardware overhead of translating the more complex x 86 architecture became negligible.
- In low-end applications, such as cell phones, the cost in power and silicon area of the x 86-translation overhead helped lead to a RISC architecture, ARM, becoming dominant.
- The combination of architectural and organizational enhancements led to 17 years of sustained growth in performance at an annual rate of over 50%—a rate that is unprecedented in the computer industry.
- Second,Dramatic improvement in cost-performance led to new classes of computers. Personal computers and workstations emerged in the 1980s with the availability of the microprocessor
- Third, improvement of semiconductor manufacturing as predicted by Moore's law has led to the dominance of microprocessor-based computers across the entire range of computer design.
- The preceding hardware innovations led to a renaissance in computer design, which emphasized both architectural innovation and efficient use of technology improvements.
- the microprocessor industry to use multiple efficient processors or cores instead of a single inefficient processor. Indeed, in 2004 Intel canceled its high-performance uniprocessor projects and joined others in declaring that the road to higher performance would be via multiple processors per chip rather than via faster uniprocessors.

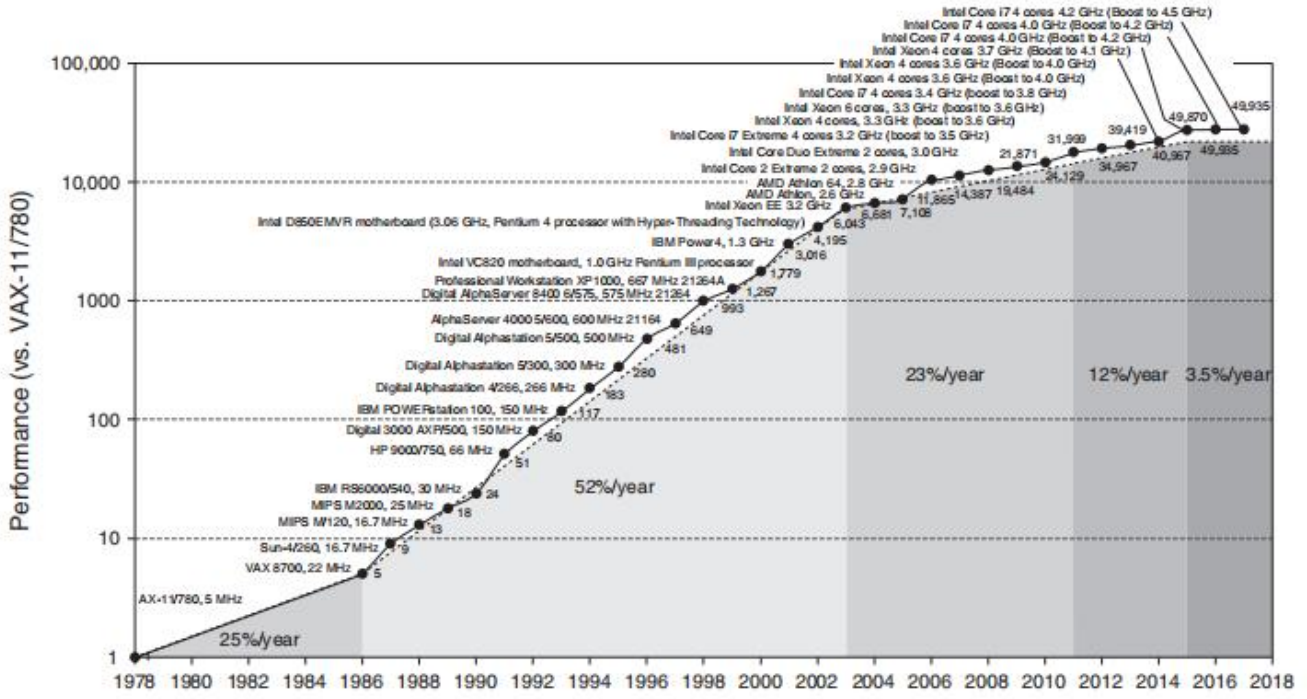


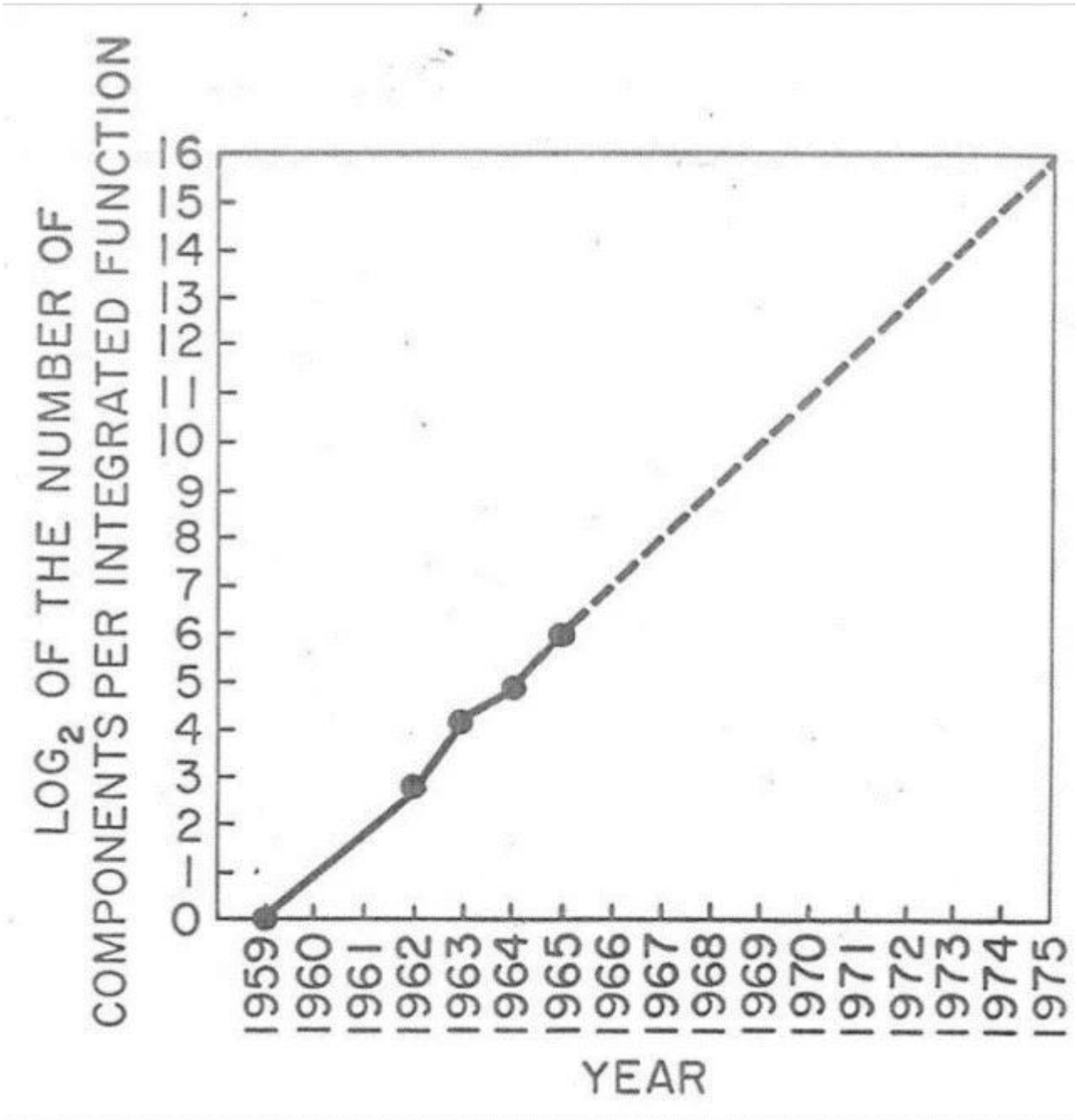
Figure 1.1 Growth in processor performance over 40 years. This chart plots program performance relative to the VAX 11/780 as measured by the SPEC integer benchmarks (see Section 1.8). Prior to the mid-1980s, growth in processor performance was largely technology-driven and averaged about 22% per year, or doubling performance every 3.5 years. The increase in growth to about 52% starting in 1986, or doubling every 2 years, is attributable to more advanced architectural and organizational ideas typified in RISC architectures. By 2003 this growth led to a difference in performance of an approximate factor of 25 versus the performance that would have occurred if it had continued at the 22% rate. In 2003 the limits of power due to the end of Dennard scaling and the available instruction-level parallelism slowed uniprocessor performance to 23% per year until 2011, or doubling every 3.5 years. (The fastest SPECintbase performance since 2007 has had automatic parallelization turned on, so uniprocessor speed is harder to gauge. These results are limited to single-chip systems with usually four cores per chip.) From 2011 to 2015, the annual improvement was less than 12%, or doubling every 8 years in part due to the limits of parallelism of Amdahl's Law. Since 2015, with the end of Moore's Law, improvement has been just 3.5% per year, or doubling every 20 years! Performance for floating-point-oriented calculations follows the same trends, but typically has 1% to 2% higher annual growth in each shaded region. Figure 1.11 on page 27 shows the improvement in clock rates for these same eras. Because SPEC has changed over the years, performance of newer machines is estimated by a scaling factor that relates the performance for different versions of SPEC: SPEC89, SPEC92, SPEC95, SPEC2000, and SPEC2006. There are too few results for SPEC2017 to plot yet.

3. Moore's Law

- Moore's law is a term used to refer to the observation made by Gordon Moore in 1965 that the number of transistors in a dense integrated circuit (IC) doubles about every two years.
- it was an observation by Gordon Moore in 1965 while he was working at Fairchild Semiconductor: the number of transistors on a microchip (as they were called in 1965) doubled about every year.
- Moore went on to co-found Intel Corporation and his observation became the driving force behind the semiconductor technology revolution at Intel and elsewhere.

How Does Moore's Law Work?

- Moore's law is based on empirical observations made by Moore. The doubling every year of the number of transistors on a microchip was extrapolated from observed data.
- Over time, the details of Moore's law were amended to better reflect actual growth of transistor density. The doubling interval was first increased to two years and then decreased to about 18 months. The exponential nature of Moore's law continued, however, creating decades of significant opportunity for the semiconductor industry. The true exponential nature of Moore's law is illustrated by the figure below.
- A straight-line plot of the logarithm of a function indicates an exponential growth of that function.



4. Classes of Parallelism and Parallel Architectures

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

1. Data-level parallelism (DLP) arises because there are many data items that can be operated on at the same time.
2. Task-level parallelism (TLP) arises because tasks of work are created that can operate independently and largely in parallel.

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

1. Instruction-level parallelism exploits data-level parallelism at modest levels with compiler help using ideas like pipelining and at medium levels using ideas like speculative execution.
2. Vector architectures, graphic processor units (GPUs), and multimedia instruction sets exploit data-level parallelism by applying a single instruction to a collection of data in parallel.
3. Thread-level parallelism exploits either data-level parallelism or task-level parallelism in a tightly coupled hardware model that allows for interaction between parallel threads.
4. Request-level parallelism exploits parallelism among largely decoupled tasks specified by the programmer or the operating system.

When Flynn (1966) studied the parallel computing efforts in the 1960s, he found a simple classification whose abbreviations we still use today. They target data-level parallelism and task-level parallelism. 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 four categories:
1. Single instruction stream, single data stream (SISD)—This category is the uniprocessor. The programmer thinks of it as the standard sequential computer, but it can exploit ILP.

2. Single instruction stream, multiple data streams (SIMD)—The same instruction is executed by multiple processors using different data streams. SIMD computers exploit data-level parallelism by applying the same operations to multiple items of data in parallel. Each processor has its own data memory (hence, the MD of SIMD), but there is a single instruction memory and control processor, which fetches and dispatches instructions.

3. Multiple instruction streams, single data stream (MISD)—No commercial multiprocessor of this type has been built to date, but it rounds out this simple classification.

4. Multiple instruction streams, multiple data streams (MIMD)—Each processor fetches its own instructions and operates on its own data, and it targets task-level parallelism. In general, MIMD is more flexible than SIMD and thus more generally applicable, but it is inherently more expensive than SIMD. For example,
MIMD computers can also exploit data-level parallelism, although the overhead is likely to be higher than would be seen in an SIMD computer. This overhead means that grain size must be sufficiently large to exploit the parallelism efficiently.

Instruction Set Architecture: The Myopic View of Computer Architecture

- The ISA serves as the boundary between the software and hardware.
- This quick review of ISA will use examples from 80x86, ARMv8, and RISC-V to illustrate the seven dimensions of an ISA.
- The most popular RISC processors come from ARM (Advanced RISC Machine).
- In addition to a full software stack (compilers, operating systems, and simulators), there are several RISC-V implementations freely available for use in custom chips or in field-programmable gate arrays.
- The integer core ISA of RISC-V as the example ISA

1. Class of ISA—Nearly all ISAs today are classified as general-purpose register architectures, where the operands are either registers or memory locations. The 80x86 has 16 general-purpose registers and 16 that can hold floating-point data, while RISC-V has 32 general-purpose and 32 floating-point registers.

Register	Name	Use	Saver
x0	zero	The constant value 0	N.A.
x1	ra	Return address	Caller
x2	sp	Stack pointer	Callee
x3	gp	Global pointer	–
x4	tp	Thread pointer	–
x5-x7	t0-t2	Temporaries	Caller
x8	s0/fp	Saved register/frame pointer	Callee
x9	s1	Saved register	Callee
x10-x11	a0-a1	Function arguments/return values	Caller
x12-x17	a2-a7	Function arguments	Caller
x18-x27	s2-s11	Saved registers	Callee
x28-x31	t3-t6	Temporaries	Caller
f0-f7	ft0-ft7	FP temporaries	Caller
f8-f9	fs0-fs1	FP saved registers	Callee
f10-f11	fa0-fa1	FP function arguments/return values	Caller
f12-f17	fa2-fa7	FP function arguments	Caller
f18-f27	fs2-fs11	FP saved registers	Callee
f28-f31	ft8-ft11	FP temporaries	Caller

Figure 1.4 RISC-V registers, names, usage, and calling conventions. In addition to the 32 general-purpose registers (x0–x31), RISC-V has 32 floating-point registers (f0–f31) that can hold either a 32-bit single-precision number or a 64-bit double-precision number. The registers that are preserved across a procedure call are labeled "Callee" saved.

2. Memory addressing—Virtually all desktop and server computers, including the 80x86, ARMv8, and RISC-V, use byte addressing to access memory operands. Some architectures, like ARMv8, require that objects must be aligned. An access to an object of size s bytes at byte address A is aligned if A mod s=0. The 80x86 and RISC-V do not require alignment, but accesses are generally faster if operands are aligned.

3. Addressing modes—In addition to specifying registers and constant operands, addressing modes specify the address of a memory object. RISC-V addressing modes are Register, Immediate (for constants), and Displacement, where a constant offset is added to a register to form the memory address.The 80x86 supports those three modes, plus three variations of displacement: no register (absolute), two registers (based indexed with displacement), and two registers where

one register is multiplied by the size of the operand in bytes (based with scaled index and displacement).

4. Types and sizes of operands—Like most ISAs, 80x86, ARMv8, and RISC-V support operand sizes of 8-bit (ASCII character), 16-bit (Unicode character or half word), 32-bit (integer or word), 64-bit (double word or long integer), and IEEE 754 floating point in 32-bit (single precision) and 64-bit (double precision). The 80x86 also supports 80-bit floating point (extended double precision).

5. Operations—The general categories of operations are data transfer, arithmetic logical, control (discussed next), and floating point. RISC-V is a simple and easy-to-pipeline instruction set architecture, and it is representative of the RISC architectures being used in 2017.

6. Control flow instructions—Virtually all ISAs, including these three, support conditional branches, unconditional jumps, procedure calls, and returns. All three use PC-relative addressing, where the branch address is specified by an address field that is added to the PC. There are some small differences. RISC-V conditional branches (BE, BNE, etc.) test the contents of registers, and the 80x86 and ARMv8 branches test condition code bits set as side effects of arithmetic/logic operations. The ARMv8 and RISC-V procedure call places the return address in a register, whereas the 80x86 call (CALLF) places the return address on a stack in memory.

7. Encoding an ISA—There are two basic choices on encoding: fixed length and variable length. All ARMv8 and RISC-V instructions are 32 bits long, which simplifies instruction decoding. Figure 1.7 shows the RISC-V instruction formats. The 80x86 encoding is variable length, ranging from 1 to 18 bytes. Variable-length instructions can take less space than fixed-length instructions, so a program compiled for the 80x86 is usually smaller than the same program compiled for RISC-V.

Trends in Technology

If an instruction set architecture is to prevail, it must be designed to survive rapid changes in Computer technology.To plan for the evolution of a computer, the designer must be aware of rapid changes in implementation technology. Five implementation technologies, which change at a dramatic pace, are critical to modern implementations:

- **Integrated circuit logic technology—**
  Historically, transistor density increased by about 35% per year, quadrupling somewhat over four years. Increases in die size are less predictable and slower, ranging from 10% to 20% per year. The combined effect was a traditional growth rate in transistor count on a chip of about 40%–55% per year, or doubling every 18–24 months. This trend is popularly known as **Moore's Law**. Device speed scales more slowly, Moore's Law is no more.

- **Semiconductor DRAM (dynamic random-access memory)—**
  This technology is the foundation of main memory The growth of DRAM has slowed dramatically, from quadrupling every three years as in the past. The 8-gigabit DRAM was shipping in 2014, but the 16-gigabit DRAM won't reach that state until 2019, and it looks like there will be no 32-gigabit DRAM.

- **Semiconductor Flash (electrically erasable programmable read-only memory)—**
  This nonvolatile semiconductor memory is the standard storage device in PMDs, and its rapidly increasing popularity has fueled its rapid growth rate in capacity. In recent years, the capacity per Flash chip increased by about 50%–60% per year, doubling roughly every 2 years. Currently, Flash memory is 8–10 times cheaper per bit than DRAM.

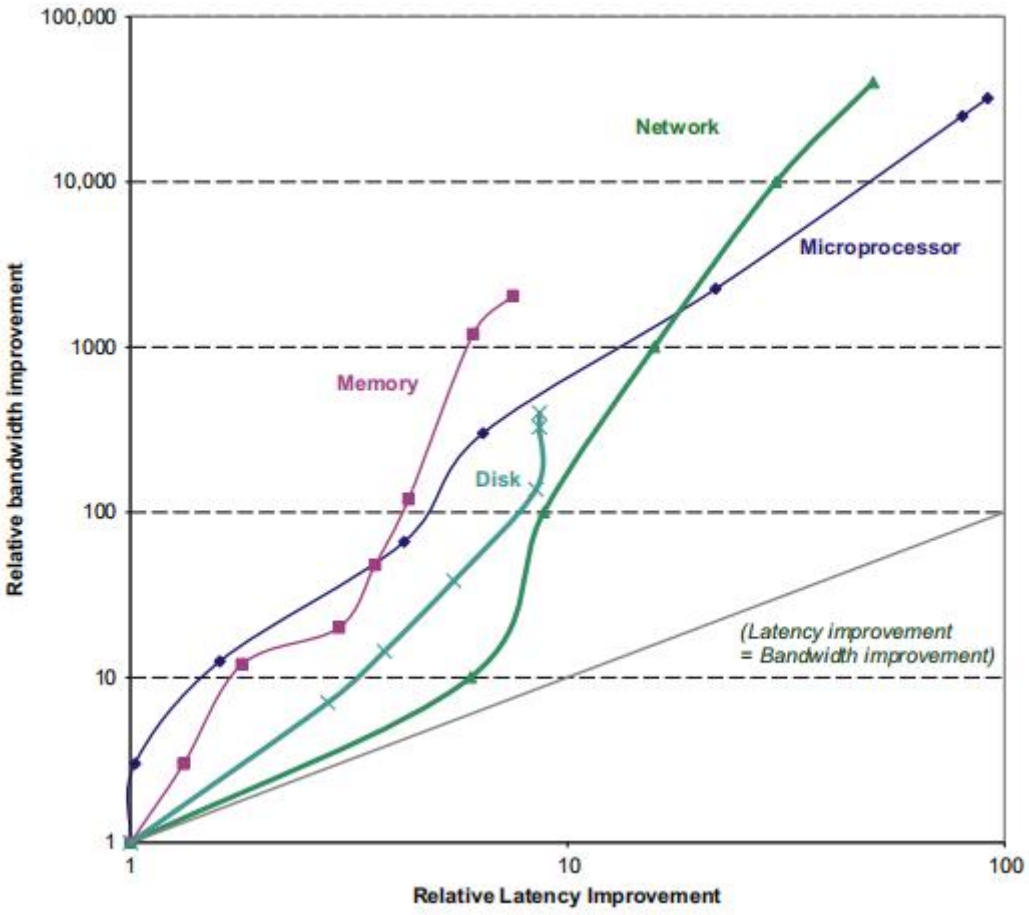
- **Magnetic disk technology—**
  Prior to 1990, density increased by about 30% per year, doubling in three years. It rose to 60% per year thereafter, and increased to 100% per year in 1996. Between 2004 and 2011, it dropped back to about 40% per year, or doubled every two years. Recently, disk improvement has slowed to less than 5% per year. One way to increase disk capacity is to add more platters at the same areal density, but there are already seven platters within the one-inch depth of the 3.5-inch form factor disks. There is room for at most one or two more platters.HAMR is the last chance for continued improvement in areal density of hard disk drives, which are now 8–10 times cheaper per bit than Flash and 200–300 times cheaper per bit than DRAM. This technology is central to server- and warehouse-scale storage.

- **Network technology—**
  Network performance depends both on the performance of switches and on the performance of the transmission system.

  These rapidly changing technologies shape the design of a computer that, with speed and technology enhancements, may have a lifetime of 3–5 years. Key technologies such as Flash change sufficiently that the designer must plan for these changes.

Performance Trends: Bandwidth Over Latency

Performance is the primary differentiator for microprocessors and networks, so they have seen the greatest gains: 32,000–40,000-X in bandwidth and 50-90-X in latency. Capacity is generally more important than performance for memory and disks, so capacity has improved more, yet bandwidth advances of 400–2400-X are still much greater than gains in latency of 8–9-X.
A simple rule of thumb is that bandwidth grows by at least the square of the improvement in latency. Computer designers should plan accordingly.



Log-log plot of bandwidth and latency milestones

Trends in Power and Energy in Integrated Circuits

Today, energy is the biggest challenge facing the computer designer for nearly every class of computer. 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.

Power and Energy: A Systems Perspective

- **First, what is the maximum power a processor ever requires?** Meeting this demand can be important to ensuring correct operation. For example, if a processor attempts to draw more power than a power-supply system can provide (by drawing more current than the system can supply), the result is typically a voltage drop, which can cause devices to malfunction. Modern processors can vary widely in power consumption with high peak currents; hence they provide voltage indexing methods that allow the processor to slow down and regulate voltage within a wider margin.
- **Second, what is the sustained power consumption?** This metric is widely called the thermal design power (TDP) because it determines the cooling requirement. TDP is neither peak power, which is often 1.5 times higher, nor is it the actual average power that will be consumed during a given computation, which is likely to be lower still. A typical power supply for a system is typically

sized to exceed the TDP, and a cooling system is usually designed to match or exceed TDP. Modern processors provide two features to assist in managing heat, since the highest power (and hence heat and temperature rise) can exceed the long-term average specified by the TDP. First, as the thermal temperature approaches the junction temperature limit, circuitry lowers the clock rate, thereby reducing power. Should this technique not be successful, a second thermal overload trap is activated to power down the chip.

- **The third factor that designers and users need to consider is energy and energy efficiency**. Recall that power is simply energy per unit time: 1 watt=1 joule per second. Which metric is the right one for comparing processors: energy or power? In general, energy is always a better metric because it is tied to a specific task and he time required for that task. In particular, the energy to complete a workload is equal to the average power times the execution time for the workload.

Conclusion-

Thus, if explain about which of two processors is more efficient for a given task, need to compare energy consumption (not power) for executing the task. For example, processor A may have a 20% higher average power consumption than processor B, but if A executes the task in only 70% of the time needed by B, its energy consumption will be 1.2* 0.7=0.84, which is clearly better.

Energy and Power Within a Microprocessor

For CMOS chips, the traditional primary energy consumption has been in switching transistors, also 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:

$$\text{Energy}_{\text{dynamic}} \propto \text{Capacitive load} \times \text{Voltage}^2$$

This equation is the energy of pulse of the logic transition of $0 \rightarrow 1 \rightarrow 0$ or $1 \rightarrow 0 \rightarrow 1$. The energy of a single transition ($0 \rightarrow 1$ or $1 \rightarrow 0$) is then:

$$\text{Energy}_{\text{dynamic}} \propto 1/2 \times \text{Capacitive load} \times \text{Voltage}^2$$

The power required per transistor is just the product of the energy of a transition multiplied by the frequency of transitions:

$$\text{Power}_{\text{dynamic}} \propto 1/2 \times \text{Capacitive load} \times \text{Voltage}^2 \times \text{Frequency switched}$$

For a fixed task, slowing clock rate reduces power, but not energy.

Clearly, dynamic power and energy are greatly reduced by lowering the voltage, so voltages have dropped from 5 V to just under 1 V in 20 years. The capacitive load is a function of the number of transistors connected to an output and the technology, which determines the capacitance of the wires and the transistors.

Example

Some microprocessors today are designed to have adjustable voltage, so a 15% reduction in voltage may result in a 15% reduction in frequency. What would be the impact on dynamic energy and on dynamic power?

Answer

Because the capacitance is unchanged, the answer for energy is the ratio of the voltages

$$\frac{\text{Energy}_{\text{new}}}{\text{Energy}_{\text{old}}} = \frac{(\text{Voltage} \times 0.85)^2}{\text{Voltage}^2} = 0.85^2 = 0.72$$

which reduces energy to about 72% of the original. For power, we add the ratio of the frequencies

$$\frac{\text{Power}_{new}}{\text{Power}_{old}} = 0.72 \times \frac{(\text{Frequency switched} \times 0.85)}{\text{Frequency switched}} = 0.61$$

shrinking power to about 61% of the original.

Trends in Cost

Although costs tend to be less important in some computer designs—specifically supercomputers—cost-sensitive designs are of growing significance.

This section discusses the major factors that influence the cost of a computer and how these factors are changing over time.

The Impact of Time, Volume, and Commoditization

- The cost of a manufactured computer component decreases over **time** even without significant improvements in the basic implementation technology.
- The underlying principle that drives costs down is the learning curve—manufacturing costs decrease over time.
- The learning curve itself is best measured by change in yield—the percentage of manufactured devices that survives the testing procedure.
- The price per megabyte of DRAM has dropped over the long term. Since DRAMs tend to be priced in close relationship to cost—except for periods when there is a shortage or an oversupply—price and cost of DRAM track closely.
- Microprocessor prices also drop over time, but because they are less standardized than DRAMs, the relationship between price and cost is more complex.
- **Volume** is a second key factor in determining cost. Increasing volumes affect cost in several ways.
- First, they decrease the time needed to get through the learning curve, which is partly proportional to the number of systems (or chips) manufactured.
- Second, volume decreases cost because it increases purchasing and manufacturing efficiency.
- volume decreases the amount of development costs that must be amortized by each computer, thus allowing cost and selling price to be closer and still make a profit.
- **Commodities** are products that are sold by multiple vendors in large volumes and are essentially identical.
- Virtually all the products sold on the shelves of grocery stores are commodities, as are standard DRAMs, Flash memory, monitors, and keyboards.
- Many vendors ship virtually identical products, the market is highly competitive.
- Of course, this competition decreases the gap between cost and selling price, but it also decreases cost.
- Reductions occur because a commodity market has both volume and a clear product definition, which allows multiple suppliers to compete in building components for the commodity product.
- As a result, the over all product cost is lower because of the competition among the suppliers of the components and the volume efficiencies the suppliers can achieve.

Cost of an Integrated Circuit

- standard parts—disks, Flash memory, DRAMs, and so on—are becoming a significant portion of any system's cost, integrated circuit costs are becoming a greater portion of the cost that varies between computers, especially in the high-volume, cost sensitive portion of the market.
- The costs of integrated circuits have dropped exponentially, the basic process of silicon manufacture is unchanged: A wafer is still tested and chopped into dies that are packaged. Therefore the cost of a packaged integrated circuit is-

$$\text{Cost of integrated circuit} = \frac{\text{Cost of die} + \text{Cost of testing die} + \text{Cost of packaging and final test}}{\text{Final test yield}}$$

- Learning how to predict the number of good chips per wafer requires first learning how many dies fit on a wafer and then learning how to predict the percentage of those that will work. From there it is simple to predict cost:-

$$\text{Cost of die} = \frac{\text{Cost of wafer}}{\text{Dies per wafer} \times \text{Die yield}}$$

- The number of dies per wafer is approximately the area of the wafer divided by the area of the die. It can be more accurately estimated by

$$\text{Dies per wafer} = \frac{\pi \times (\text{Wafer diameter}/2)^2}{\text{Die area}} - \frac{\pi \times \text{Wafer diameter}}{\sqrt{2 \times \text{Die area}}}$$

- The first term is the ratio of wafer area (πr2) to die area. The second compensates for the "square peg in a round hole" problem—rectangular dies near the periphery of round wafers. Dividing the circumference (πd) by the diagonal of a square die is approximately the number of dies along the edge.

Example Find the number of dies per 300 mm (30 cm) wafer for a die that is 1.5 cm on a side and for a die that is 1.0 cm on a side.

Answer When die area is 2.25 cm^2:

$$\text{Dies per wafer} = \frac{\pi \times (30/2)^2}{2.25} - \frac{\pi \times 30}{\sqrt{2 \times 2.25}} = \frac{706.9}{2.25} - \frac{94.2}{2.12} = 270$$

Because the area of the larger die is 2.25 times bigger, there are roughly 2.25 as many smaller dies per wafer:

$$\text{Dies per wafer} = \frac{\pi \times (30/2)^2}{1.00} - \frac{\pi \times 30}{\sqrt{2 \times 1.00}} = \frac{706.9}{1.00} - \frac{94.2}{1.41} = 640$$

This formula gives only the maximum number of dies per wafer. The critical question is: What is the fraction of good dies on a wafer, or the die yield? A simple model of integrated circuit yield, which assumes that defects are randomly distributed over the wafer and that yield is inversely proportional to the complexity of the fabrication process, leads to the following:

$$\text{Die yield} = \text{Wafer yield} \times 1/(1 + \text{Defects per unit area} \times \text{Die area})^N$$

Example Find the die yield for dies that are 1.5 cm on a side and 1.0 cm on a side, assuming a defect density of 0.047 per cm^2 and N is 12.

Answer The total die areas are 2.25 and 1.00 cm^2. For the larger die, the yield is

$$\text{Die yield} = 1/(1 + 0.047 \times 2.25)^{12} \times 270 = 120$$

For the smaller die, the yield is

$$\text{Die yield} = 1/(1 + 0.047 \times 1.00)^{12} \times 640 = 444$$

The bottom line is the number of good dies per wafer. Less than half of all the large dies are good, but nearly 70% of the small dies are good.

Performance Metrics and Evaluation

- Hardware performance is key to the effectiveness of the entire system.
- Performance has to be measured and compared
  - Evaluate various design and technological approaches
- Different types of applications:
  - Different performance metrics may be appropriate
  - Different aspects of a computer system may be most significant
- Factors that affect performance
  - Instruction use and implementation, memory hierarchy, I/O handling.

Defining Performance

- Performance means different things to different people
- Analogy from the airline industry:
  - Cruising speed (How fast)
  - Flight range (How far)
  - Passengers (How many)

Performance Metrics

- **Response (execution) time:**
  - Time between the start and completion of a task
  - Measures user perception of the system speed
  - Common in reactive and time critical systems, single-user computer, etc.
- **Throughput:**
  - Total number of tasks done in a given time
  - Most relevant to batch processing (billing, credit card processing, etc.)
  - Mainly used for input/output systems (disk access, printer, etc.)

Relate the performance of two different computers, say, X and Y. The phrase "X is faster than Y" is used here to mean that the response time or execution time is lower on X than on Y for the given task.
In particular, "X is n times as fast as Y" will mean -

$$\frac{\text{Execution time}_Y}{\text{Execution time}_X} = n$$

Since execution time is the reciprocal of performance, the following relationship holds:

$$n = \frac{\text{Execution time}_Y}{\text{Execution time}_X} = \frac{\frac{1}{\text{Performance}_Y}}{\frac{1}{\text{Performance}_X}} = \frac{\text{Performance}_X}{\text{Performance}_Y}$$

"the throughput of X is 1.3 times as fast as Y" signifies here that the number of tasks completed per unit time on computer X is 1.3 times the number completed on Y.

- Time is not always the metric quoted in comparing the performance of computers.
- the only consistent and reliable measure of performance is the execution time of real programs, and that all proposed alternatives to time as the metric or to real programs as the items measured have eventually led to misleading claims or even mistakes in computer design.
- Even execution time can be defined in different ways depending on what count.

- The most straightforward definition of time is called wall-clock time, response time, or elapsed time, which is the latency to complete a task, including storage accesses, memory accesses, input/output activities, operating system overhead—everything.

Benchmarks

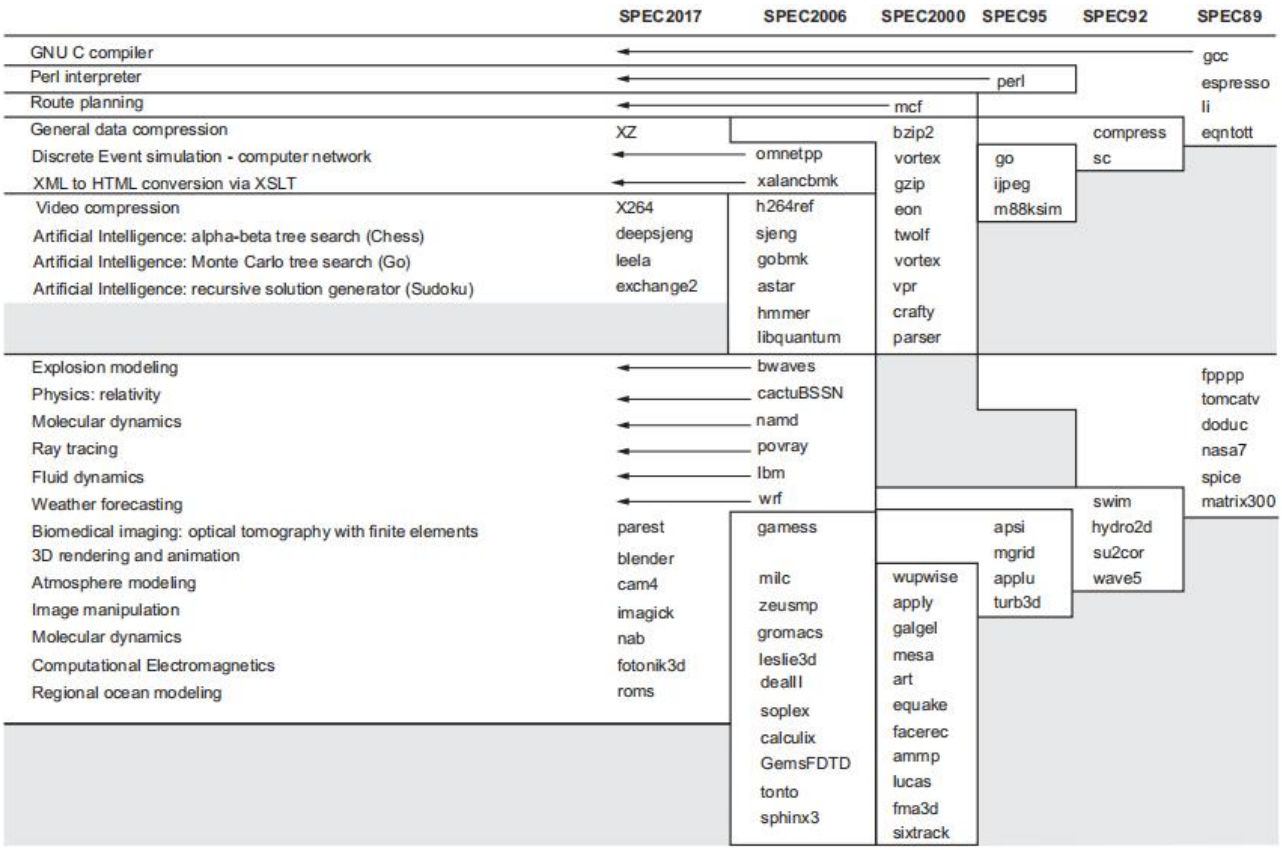
- The best choice of benchmarks to measure performance is real applications, such as Google Translate.
- Attempts at running programs that are much simpler than a real application have led to performance pitfalls.

Kernels, which are small, key pieces of real applications.

- Toy programs, which are 100-line programs from beginning programming assignments, such as Quicksort.
- Synthetic benchmarks, which are fake programs invented to try to match the profile and behavior of real applications, such as Dhrystone.
- One way to improve the performance of a benchmark has been with benchmark-specific compiler flags; these flags often caused transformations that would be illegal on many programs or would slow down performance on others.
- To restrict this process and increase the significance of the results, benchmark developers typically require the vendor to use one compiler and one set of flags for all the programs in the same language (such as C++ or C).
- In addition to the question of compiler flags, another question is whether source code modifications are allowed. There are three different approaches to addressing this question:
  1. No source code modifications are allowed.
  2. Source code modifications are allowed but are essentially impossible. For example, database benchmarks rely on standard database programs that are tens of millions of lines of code. The database companies are highly unlikely to make changes to enhance the performance for one particular computer.
  3. Source modifications are allowed, as long as the altered version produces the same output.
- The key issue that benchmark designers face in deciding to allow modification of the source is whether such modifications will reflect real practice and provide useful insight to users, or whether these changes simply reduce the accuracy of the benchmarks as predictors of real performance.
- The goal of a benchmark suite is that it will characterize the real relative performance of two computers, particularly for programs not in the suite that customers are likely to run.
- One of the most successful attempts to create standardized benchmark application suites has been the SPEC (Standard Performance Evaluation Corporation), which had its roots in efforts in the late 1980s to deliver better benchmarks for workstations.
- Just as the computer industry has evolved over time, so has the need for different benchmark suites, and there are now SPEC benchmarks to cover many application classes.

Desktop Benchmarks

- Desktop benchmarks divide into two broad classes: processor-intensive benchmarks and graphics-intensive benchmarks, although many graphics benchmarks include intensive processor activity.
- SPEC originally created a benchmark set focusing on processor performance (initially called SPEC89), which has evolved into its sixth generation.



SPEC2017 programs and the evolution of the SPEC benchmarks over time, with integer programs above the line and floating point programs below the line.

- SPEC benchmarks are real programs modified to be portable and to minimize the effect of I/O on performance.
- The integer benchmarks vary from part of a C compiler to a go program to a video compression.
- The floating-point benchmarks include molecular dynamics, ray tracing, and weather forecasting.
- The SPEC CPU suite is useful for processor bench marking for both desktop systems and single-processor servers.

Server Benchmark

- Servers have multiple functions, so are there multiple types of benchmarks.
- The simplest benchmark is perhaps a processor throughput-oriented benchmark.
- SPEC CPU2017 uses the SPEC CPU benchmarks to construct a simple throughput benchmark where the processing rate of a multiprocessor can be measured by running multiple copies (usually as many as there are processors) of each SPEC CPU benchmark and converting the CPU time into a rate.
- This leads to a measurement called the **SPECrate**, and it is a measure of request-level parallelism
- To measure thread-level parallelism, SPEC offers what they call high performance computing benchmarks around OpenMP and MPI as well as for accelerators such as GPUs
- Other than SPECrate, most server applications and benchmarks have significant I/O activity arising from either storage or network traffic, including benchmarks for file server systems, for web servers, and for database and transaction processing systems.
- Transaction-processing (TP) benchmarks measure the ability of a system to handle transactions that consist of database accesses and updates.

- Airline reservation systems and bank ATM systems are typical simple examples of TP; more sophisticated TP systems involve complex databases and decision-making.
- In the mid-1980s, a group of concerned engineers formed the vendor-independent Transaction Processing Council (TPC) to try to create realistic and fair benchmarks for TP.
- The first TPC benchmark, TPC-A, was published in 1985 and has since been replaced and enhanced by several different benchmarks.
- TPC-C, initially created in 1992, simulates a complex query environment.
- TPC-H models ad hoc decision support—the queries are unrelated and knowledge of past queries cannot be used to optimize future queries.
- The TPC-DI benchmark, a new data integration (DI) task also known as ETL, is an important part of data warehousing.
- TPC-E is an online transaction processing (OLTP) workload that simulates a brokerage firm's customer accounts.
- **All the TPC benchmarks measure performance in transactions per second. In addition, they include a response time requirement so that throughput performance is measured only when the response time limit is met.**
- The system cost for a benchmark system must be included as well to allow accurate comparisons of cost-performance.

Performance Summarizing Methods

Average execution time: calculate the average time for each execution and then calculate the speedup. We can not simply average the values of the speedups. This is the commonest method we normally use for summarizing performance

Geometric mean: if we want to calculate the mean value by the speedups, we have to use the geometric mean. We are going to use the geometric mean only if we don't know the actual time for each application. The geometric mean is defined by,

$$\left(\prod_{i=1}^{n} a_i\right)^{1/n} = \sqrt[n]{a_1\, a_2\, a_3 \ldots . a_n}$$

Let's see an example. Where AVG represents average execution time and GEO means for geometric mean,

	COMP_X	COMP_Y	SPEEDUP
APP A	9	18	2
APP B	10	7	0.7
APP C	5	11	2.2
AVG	8	12	1.5
GEO	7.66	11.15	1.46

Iron Law of Performance

(1) CPU Time

In this series, we are going to focus mainly on the processor. When we say processor, we are always meaning the CPU (central processing unit). The CPU time or the processor time is defined as the time for a processor to run a program.

(2) Iron Law of Performance

Actually, the CPU time can be calculated by,

$$CPU\ Time = \#\ of\ instructions\ per\ program \cdot cycles\ per\ instruction \cdot clock\ cycle\ time$$

This can be proved by the following expression,

$$CPU\ Time = \frac{\#\ of\ instructions}{program} \cdot \frac{\#\ of\ cycles}{\#\ of\ instructions} \cdot \frac{Seconds}{\#\ of\ cycles}$$

The three components of the CPU time allow us to think about different aspects of computer architecture. This is called the Iron Law of performance. We can also write this expression to,

$$CPU\ Time = IC \cdot CPI \cdot CCT$$

where,

- IC means instructions count
- CPI means cycles per instruction
- CCT means clock cycle time (which is also the reciprocal of the cycle clock)

(3) Components of the CPU Time And Computer Architecture

- # of instructions per program: generally affected by the algorithm we use and the compiler we use. Also, the instruction set can impact how many instructions we need for a program
- # of cycles per instruction: affected by the instruction set (a simple set will reduce the cycles to do) and the processor design
- clock cycle time: affected by the processor design, the circuit design, and the transistor physics

(4) Iron Law for Unequal Instruction Times

- The iron law of performance can be easily applied to the instructions that have constant cycles, but how can we apply this law to the unequal instruction times. This is a simple mathematic problem and the final processor time can be calculated by,

$$CPU\ Time = CCT \cdot \sum_{i} IC_i \times CPI_i$$

Amdahl's Law

The performance gain that can be obtained by improving some portion of a computer can be calculated using Amdahl's Law. Amdahl's Law states that the performance improvement to be gained from using some faster mode of execution is limited by the fraction of the time the faster mode can be used.

(1) Definition of the Amdahl's Law

- Another law we may use a lot is called the Amdahl's Law, especially when we are about to speed up only part of the program or only some instructions. Basically, when are have a speed-up task but it doesn't apply to the entire program and we want to know what is the overall speed up on the entire program.
- The fraction enhanced is defined as the percentage of the original execution TIME (not instructions count or something else) that is affected by our enhancement and this value is denoted as FRAC_enh.

- By Amdahl's Law, the overall speedup can be calculated by,

$$\text{Speedup} = \frac{\text{Performance for entire task using the enhancement when possible}}{\text{Performance for entire task without using the enhancement}}$$

Alternatively,

$$\text{Speedup} = \frac{\text{Execution time for entire task without using the enhancement}}{\text{Execution time for entire task using the enhancement when possible}}$$

$$Overall\ Speedup = \frac{1}{(1 - FRAC_{enh}) + \frac{FRAC_{enh}}{SPEEDUP_{enh}}}$$

Amdahl's Law gives us a quick way to find the speedup from some enhancement, which depends on two factors:

1. The fraction of the computation time in the original computer that can be converted to take advantage of the enhancement—For example, if 40 seconds of the execution time of a program that takes 100 seconds in total can use an enhancement, the fraction is 40/100. This value, which we call **Fraction**$_{\text{enhanced}}$, is always less than or equal to 1.

2. The improvement gained by the enhanced execution mode, that is, how much faster the task would run if the enhanced mode were used for the entire program—This value is the time of the original mode over the time of the enhanced mode. If the enhanced mode takes, say, 4 seconds for a portion of the program, while it is 40 seconds in the original mode, the improvement is 40/4 or 10. We call this value, which is always greater than 1, **Speedup**$_{\text{enhanced}}$.

The execution time using the original computer with the enhanced mode will be the time spent using the unenhanced portion of the computer plus the time spent using the enhancement:

$$\text{Execution time}_{new} = \text{Execution time}_{old} \times \left((1 - \text{Fraction}_{enhanced}) + \frac{\text{Fraction}_{enhanced}}{\text{Speedup}_{enhanced}} \right)$$

The overall speedup is the ratio of the execution times:

$$\text{Speedup}_{overall} = \frac{\text{Execution time}_{old}}{\text{Execution time}_{new}} = \frac{1}{(1 - \text{Fraction}_{enhanced}) + \frac{\text{Fraction}_{enhanced}}{\text{Speedup}_{enhanced}}}$$

(2) Amdahl's Law Implications

- So what does Amdahl's law implies? Let's think about two different enhancements. For enhancement #1, we enhance a speedup of 20 on 10% of the time. For enhancement #2, we enhance a speedup of 1.6 on 80% of the time. By Amdahl's law, the overall speedup for enhancement #1 is 1.105 and the overall speedup for enhancement #2 is 1.43 .
- This implies that if we put significant effort to speed up something that is a small part of the execution time, you will still not get a very large improvement. In contrast, if you are impacting a large part of the execution time and even if there's only a small reasonably small speedup, you will still result in a large overall speedup.

Example Suppose that we want to enhance the processor used for web serving. The new processor is 10 times faster on computation in the web serving application than the old processor. Assuming that the original processor is busy with computation 40% of the time and is waiting for I/O 60% of the time, what is the overall speedup gained by incorporating the enhancement?

Answer $\text{Fraction}_{\text{enhanced}} = 0.4;\ \text{Speedup}_{\text{enhanced}} = 10;\ \text{Speedup}_{\text{overall}} = \dfrac{1}{0.6 + \dfrac{0.4}{10}} = \dfrac{1}{0.64} \approx 1.56$

Lhadma's Law

This is jokingly called Lhadma's law because of Amdahl's law. We have known that Amdahl's law implies that we have to speed up the most common cases in order to have a significant impact on the overall speedup, while Lhadma's law tells us that we don't want to mess up with the uncommon cases too badly.

Let's see an example. Suppose we would like to have a speedup of 2 on 90% of the execution time and another speedup of 0.1 on the rest of the execution time. The overall speedup can be calculated by,

$$Overall\ Speedup = \frac{1}{\frac{10\%}{0.1} + \frac{90\%}{2}} = 0.7$$

And this result proves Lhadma's law.