Performance of Computing Systems & Its Role

Performance

- **Computer performance** is the amount of work accomplished by a computer system. The word performance in computer performance means “How well is the computer doing the work it is supposed to do?”.
- Being able to gauge the relative performance of a computer is an important but tricky task. There are a lot of factors that can affect performance.

  -Architecture

  -Hardware implementation of the architecture

  -Compiler for the architecture

  -Operating system
-

- Furthermore, we need to be able to define a measure of performance.

  - Single users on a PC $\longrightarrow$ a minimization of response time.

  -Large data $\longrightarrow$ a maximization of throughput

- To discuss performance, we need to be familiar with two terms:

  - Latency (response time) is the time between the start and completion of an event.

  - Throughput (bandwidth) is the total amount of work done in a given period of time.

Q : Do the following changes to a computer system increase throughput, decrease latency, or both?

1. Replacing the processor in a computer with a faster processor.

   Throughput increases and latency decreases (i.e. both improve).

2. Adding additional processors to a system that uses processors for separate tasks.

   Only throughput increases

Performance and execution time

- Performance has an inverse relationship to execution time.

$$Performance = \frac{1}{Execution\,Time}$$

- Comparing the performance of two machines can be accomplished by comparing execution times.

$$Performance_X > Performance_Y$$

$$\longrightarrow \quad \frac{1}{Execution_X} > \frac{1}{Execution_Y}$$

$$\longrightarrow \quad Execution_Y > Execution_X$$

Relative Performance

- Often people state that a machine X is n times faster than a machine Y.

$$\frac{Performance_X}{Performance_Y} = \frac{Execution_Y}{Execution_X} = n$$

$$\frac{\frac{1}{E_X}}{\frac{1}{E_Y}} \qquad \frac{E_Y}{E_X}$$

If computer A runs a program in 10 seconds and computer B runs the same program in 15 seconds, how much faster is A than B?

We know that A is n times as fast as B if

$$\frac{\text{Performance}_A}{\text{Performance}_B} = \frac{\text{Execution time}_B}{\text{Execution time}_A} = n$$

Thus the performance ratio is

$$\frac{15}{10} = \underline{\underline{1.5}}$$

and A is therefore 1.5 times as fast as B.

In the above example, we could also say that computer B is 1.5 times *slower than* computer A, since

$$\frac{\text{Performance}_A}{\text{Performance}_B} = 1.5$$

means that

$$\frac{\text{Performance}_A}{1.5} = \text{Performance}_B$$

Measuring Performance

- There are several ways to measure the execution time on a machine.
  - Elapsed time: total wall clock time needed to complete a task (including I/O, etc).
  - CPU time: time CPU spends actually working on behalf of the program. This does not include waiting for I/O or other running programs.

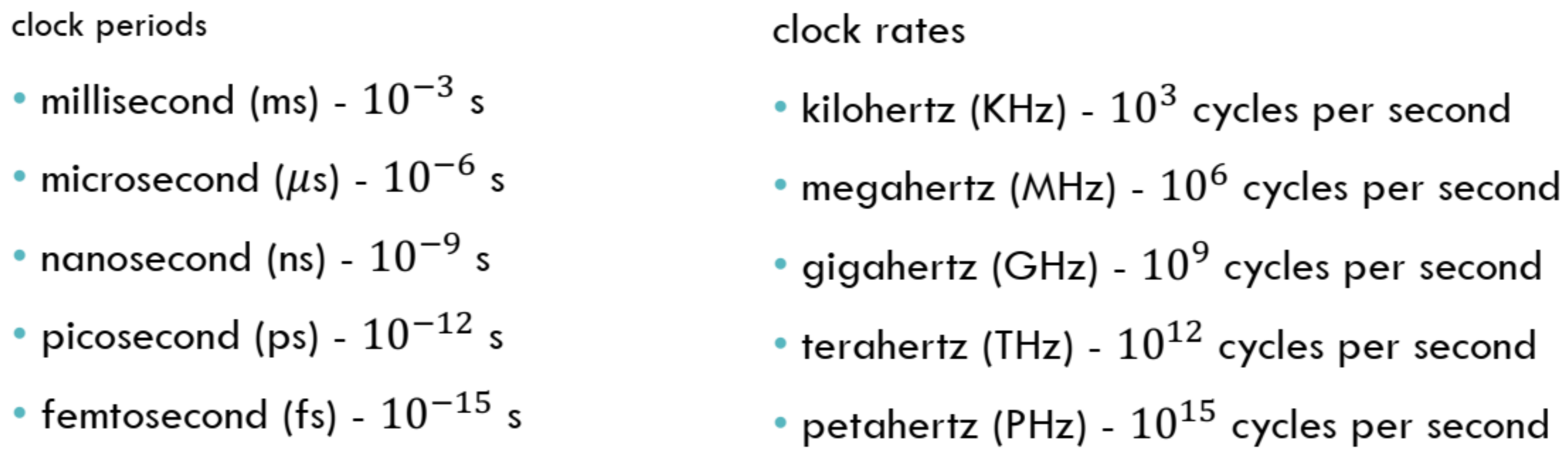
Computers are often shared, however, and a processor may work on several programs simultaneously. In such cases, the system may try to optimize throughput rather than attempt to minimize the elapsed time for one program.

  - User CPU time: CPU time spent in the program itself.
  - System CPU time: CPU time spent in the OS, performing tasks on behalf of the program.

Measuring Performance

- Sometimes, it is more useful to think about performance in metrics other than time. In particular, it is common to discuss performance in terms of how fast a computer can perform basic functions.
- Clock cycle: The basic discrete time intervals of a processor clock, which runs at a constant rate (mostly). A clock cycle, or simply a "cycle," is a single electronic pulse of a CPU. During each cycle, a CPU can perform a basic operation such as fetching an instruction, accessing memory, or writing data. Since only simple commands can be performed during each cycle, most CPU processes require multiple clock cycles.
- Clock period: the length of each clock cycle.
- Clock rate: inverse of the clock period.

- Some common prefixes for clock period and clock rate:



clock periods

- millisecond (ms) - 10^{-3} s
- microsecond (μs) - 10^{-6} s
- nanosecond (ns) - 10^{-9} s
- picosecond (ps) - 10^{-12} s
- femtosecond (fs) - 10^{-15} s

clock rates

- kilohertz (KHz) - 10^{3} cycles per second
- megahertz (MHz) - 10^{6} cycles per second
- gigahertz (GHz) - 10^{9} cycles per second
- terahertz (THz) - 10^{12} cycles per second
- petahertz (PHz) - 10^{15} cycles per second

CPU Performance

In order to determine the effect of a design change on the performance experienced by the user, we can use the following relation:

$$CPU\ Execution\ Time = CPU\ Clock\ Cycles \times Clock\ Period$$

Alternatively,

$$CPU\ Execution\ Time = \frac{CPU\ Clock\ Cycles}{Clock\ Rate}$$

Clearly, we can reduce the execution time of a program by either reducing the number of clock cycles required or the length of each clock cycle.

Instruction Performance

- Another way to think about program execution time is in terms of instruction performance. Generally, execution time is equal to the number of instructions executed multiplied by the average time per instruction.

$$CPU\ Clock\ Cycles = Instructions\ for\ a\ Program \times Average\ Clock\ Cycles\ Per\ Instruction$$

- Clock cycles per instruction (CPI), which is the average number of clock cycles each instruction takes to execute, is often abbreviated as CPI.

The above equation can be rearranged to give the following:

$$CPI = \frac{CPU\ Clock\ Cycles}{Instruction\ Count}$$

Classic CPU Performance equation

We can now write the basic equation in terms of instruction count, CPI, and clock cycle time.

$$CPU\ Time = Instruction\ Count \times CPI \times Clock\ Period$$

Alternatively,

$$CPU\ Time = \frac{Instruction\ Count \times CPI}{Clock\ Rate}$$

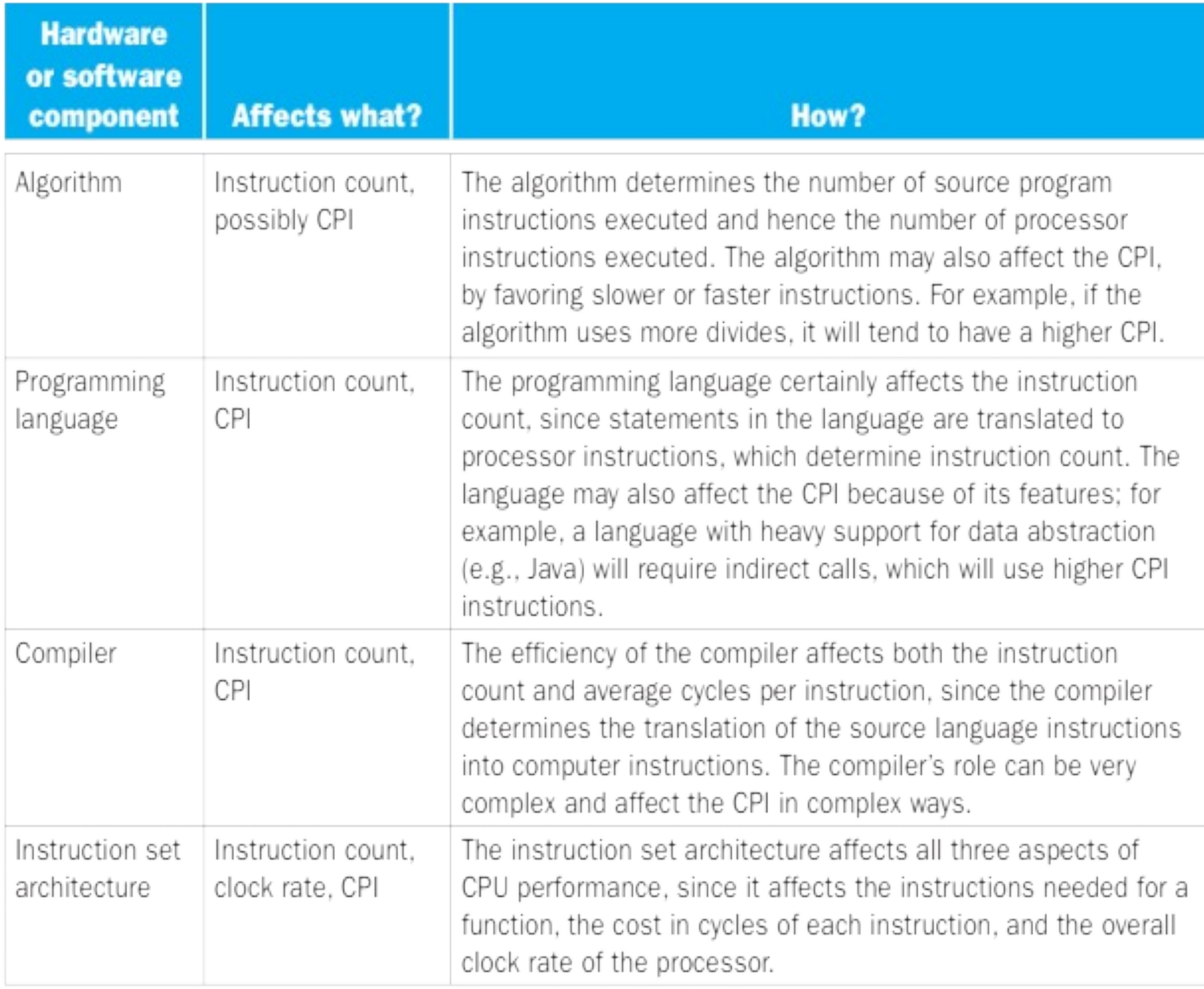
Components of Performance

The basic components of performance and how each is measured.

Component	**Units of Measure**
CPU Execution Time for a Program	Seconds for the Program
Instruction Count	Instructions Executed for the Program
Clock Cycles per Instruction	Average Number of Clock Cycles per Instruction
Clock Cycle Time (Clock Period)	Seconds per Clock Cycle

Instruction Count, CPI, and Clock Period combine to form the three important components for determining CPU execution time. *Just analyzing one is not enough!* Performance between two machines can be determined by examining non-identical components.

- The performance of a program depends on the algorithm, the language, the compiler, the architecture, and the actual hardware.



Hardware or software component	Affects what?	How?
Algorithm	Instruction count, possibly CPI	The algorithm determines the number of source program instructions executed and hence the number of processor instructions executed. The algorithm may also affect the CPI, by favoring slower or faster instructions. For example, if the algorithm uses more divides, it will tend to have a higher CPI.
Programming language	Instruction count, CPI	The programming language certainly affects the instruction count, since statements in the language are translated to processor instructions, which determine instruction count. The language may also affect the CPI because of its features; for example, a language with heavy support for data abstraction (e.g., Java) will require indirect calls, which will use higher CPI instructions.
Compiler	Instruction count, CPI	The efficiency of the compiler affects both the instruction count and average cycles per instruction, since the compiler determines the translation of the source language instructions into computer instructions. The compiler's role can be very complex and affect the CPI in complex ways.
Instruction set architecture	Instruction count, clock rate, CPI	The instruction set architecture affects all three aspects of CPU performance, since it affects the instructions needed for a function, the cost in cycles of each instruction, and the overall clock rate of the processor.

Amdahl’s law

- The execution time of the program after making the improvement is given by the following simple equation known as 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.

$$\text{Execution time after Improvement} = \frac{\text{Execution time affected by improvement}}{\text{Amount of improvement}} + \text{Execution time unaffected}$$

- Amdahl's Law depends on two factors:

    - The fraction of the time the enhancement can be exploited(fraction enhanced)

    - The improvement gained by the enhanced execution mode(speedup enhanced)

Fraction enhanced – The fraction of the computation time in the original computer that can be converted to take advantage of the enhancement. For example- if 10 seconds of the execution time of a program that takes 40 seconds in total can use an enhancement , the fraction is 10/40. This obtained value is *Fraction Enhanced*.
Fraction enhanced is always less than 1.

Speedup enhanced – 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. For example – If the enhanced mode takes, say 3 seconds for a portion of the program, while it is 6 seconds in the original mode, the improvement is 6/3. This value is Speedup enhanced.
Speedup Enhanced is always greater than 1.

- Speedup is defined as the ratio of performance for the entire task using the enhancement and performance for the entire task without using the enhancement or speedup can be defined as the ratio of execution time for the entire task without using the enhancement and execution time for the entire task using the enhancement.
- If **Pe** is the performance for entire task using the enhancement when possible, **Pw** is the performance for entire task without using the enhancement, **Ew** is the execution time for entire task without using the enhancement and **Ee** is the execution time for entire task using the enhancement when possible then,

Speedup = Pe/Pw
or
Speedup = Ew/Ee

- We can use Amdahl’s Law to estimate performance improvements when we know the time consumed for some function and its potential speedup.
- Amdahl’s Law, together with the CPU performance equation, is a handy tool for evaluating potential enhancements.

MIPS (Millions Instructions Per Second)

- One alternative to time is MIPS (million instructions per second).
- A measurement of program execution speed based on the number of millions of instructions.
- MIPS is computed as the instruction count divided by the product of the execution time and 10^6 (million)

$$\text{MIPS} = \frac{\text{Instruction count}}{\text{Execution time} \times 10^6}$$

- Since the MIPS estimation doesn’t take into consideration other components such as the computer’s I/O speed or processor engineering, it isn’t continuously a reasonable way to degree the execution of a computer.

- **Advantages :**

Since MIPS is an instruction execution rate, MIPS specifies performance inversely to execution time; faster computers have a higher MIPS rating. The good news about MIPS is that it is easy to understand, and faster computers mean bigger MIPS, which matches intuition.

- **Disadvantages**

- MIPS specifies the instruction execution rate but does not take into account the capabilities of the instructions. We cannot compare computers with different instruction sets using MIPS, since the instruction counts will certainly differ.

- MIPS varies between programs on the same computer; thus, a computer cannot have a single MIPS rating.

- Finally, and most importantly, if a new program executes more instructions but each instruction is faster, MIPS can vary independently from performance!