*Technical University of Cluj-Napoca Faculty of Computers and Automation*

**Evaluation of the facilities of multicore processors using a custom set of benchmark programs**

Cluj-Napoca

08.01.2020

Harhoi Radu

Group 30433

1. **Introduction**

A multi-core processor  is a [computer processor](https://en.wikipedia.org/wiki/Computer_processor) [integrated circuit](https://en.wikipedia.org/wiki/Integrated_circuit) with two or more separate [processing units](https://en.wikipedia.org/wiki/Central_processing_unit), called cores, each of which reads and executes [program instructions](https://en.wikipedia.org/wiki/Instruction_set), as if the computer had several processors The instructions are ordinary [CPU instructions](https://en.wikipedia.org/wiki/Instruction_set) (such as add, move data, and branch) but the single processor can run instructions on separate cores at the same time, increasing overall speed for programs that support  [multithreading](https://en.wikipedia.org/wiki/Multithreading_(computer_architecture)) or other [parallel computing](https://en.wikipedia.org/wiki/Parallel_computing) techniques.

While manufacturing technology improves, reducing the size of individual gates, physical limits of [semiconductor](https://en.wikipedia.org/wiki/Semiconductor)-based [microelectronics](https://en.wikipedia.org/wiki/Microelectronics) have become a major design concern. These physical limitations can cause significant heat dissipation and data synchronization problems. Various other methods are used to improve CPU performance. Some [instruction-level parallelism](https://en.wikipedia.org/wiki/Instruction-level_parallelism) (ILP) methods such as [superscalar](https://en.wikipedia.org/wiki/Superscalar) [pipelining](https://en.wikipedia.org/wiki/Pipelining) are suitable for many applications, but are inefficient for others that contain difficult-to-predict code. Many applications are better suited to [thread-level parallelism](https://en.wikipedia.org/wiki/Thread-level_parallelism) (TLP) methods, and multiple independent CPUs are commonly used to increase a system's overall TLP. A combination of increased available space (due to refined manufacturing processes) and the demand for increased TLP led to the development of multi-core CPUs.

Fast, inexpensive computers are now essential to numerous human endeavors. In order to assure a certain quality, it is important to measure and compare the performance of different computers.

Computer performance is the amount of useful work accomplished by a computer system. Outside of specific contexts, computer performance is estimated in terms of accuracy, efficiency and speed of executing [computer program](https://en.wikipedia.org/wiki/Computer_program) instructions. Some examples of performance metrics are:

1. **Clock frequency (Hz**)

A CPU's clock speed represents how many cycles per second it can execute. Clock speed is also referred to as clock rate, PC frequency and CPU frequency. This is measured in gigahertz, which refers to billions of pulses per second and is abbreviated as GHz.

A PC’s clock speed is an indicator of its performance and how rapidly a CPU can process data (move individual bits). A higher frequency (bigger number) suggests better performance in common tasks, such as [gaming](https://www.tomshardware.com/reviews/best-cpus,3986.html). A CPU with higher clock speed is generally better if all other factors are equal, but a mixture of clock speed, how many instructions the CPU can process per cycle (also known as instructions per clock cycle/clock, or IPC for short) and the number of [cores](https://www.tomshardware.com/news/cpu-core-definition,37658.html)the CPU has all help determine overall performance.

1. **FLOPS – floating point operations per second**

[Floating-point arithmetic](https://en.wikipedia.org/wiki/Floating-point_arithmetic) is needed for very large or very small [real numbers](https://en.wikipedia.org/wiki/Real_number), or computations that require a large dynamic range. Floating-point representation is similar to scientific notation, except everything is carried out in base two, rather than base ten. The encoding scheme stores the sign, the [exponent](https://en.wikipedia.org/wiki/Exponent) (in base two for Cray and [VAX](https://en.wikipedia.org/wiki/VAX), base two or ten for [IEEE floating point](https://en.wikipedia.org/wiki/IEEE_floating_point) formats, and base 16 for [IBM Floating Point Architecture](https://en.wikipedia.org/wiki/IBM_Floating_Point_Architecture)) and the [Significand](https://en.wikipedia.org/wiki/Significand" \o "Significand) (number after the radix point). While several similar formats are in use, the most common is [ANSI/IEEE Std. 754-1985](https://en.wikipedia.org/wiki/IEEE_754-1985). This standard defines the format for 32-bit numbers called single precision, as well as 64-bit numbers called double precision and longer numbers called extended precision (used for intermediate results). Floating-point representations can support a much wider range of values than fixed-point, with the ability to represent very small numbers and very large numbers.

1. **Memory Bandwidth (B/s)**

In computer networking, bandwidth is a measurement of bit-rate of available or consumed data communication resources, expressed in bits per second or multiples of it (bit/s, kb/s, Mb/s, Gb/s, etc.).

Bandwidth sometimes defines the net bit rate (aka. peak bit rate, information rate, or physical layer useful bit rate), channel capacity, or the maximum throughput of a logical or physical communication path in a digital communication system. For example, bandwidth tests measure the maximum throughput of a computer network. The reason for this usage is that according to Hartley's law, the maximum data rate of a physical communication link is proportional to its bandwidth in hertz, which is sometimes called frequency bandwidth, spectral bandwidth, RF bandwidth, signal bandwidth or analog bandwidth.

1. **Storage Device Speed (B/s)**

Read/write speeds are used to measure the performance of a storage device. The read speed refers to how long it takes to open a file from the device, and the write speed is how long it takes to save a file to the device. Read/write speed tests can be performed on internal and [external hard disk drives](https://www.lifewire.com/what-is-an-external-drive-2625867) as well as [storage area networks](https://www.lifewire.com/definition-of-san-818007) and [USB flash drives](https://www.lifewire.com/what-is-a-flash-drive-2625794).

Read and write speeds are typically recorded with the letters **ps** (per second) at the end of the measurement. For example, a device that has a write speed of 32 MBps means that it can record 32 MB ([megabytes](https://www.lifewire.com/what-is-a-megabit-2483412)) of data every second.

The purpose of this work is to show the difference between the computing performance of different machines by creating a set of benchmark programs to compute the metrics discussed before.

1. **Objectives**

a. Design algorithms that measure each of the parameters from the previous section

b. Run the algorithms multiple times measuring time both using c++ and assembly

c. Get the results and display them graphically

d. Compare the results and draw conclusions

1. **Theoretical fundamentals**

**3.1. Clock frequency**

For a given CPU, the clock rates are determined at the end of the manufacturing process through actual testing of each processor. Chip manufacturers publish a "maximum clock rate" specification, and they test chips before selling them to make sure they meet that specification, even when executing the most complicated instructions with the data patterns that take the longest to settle (testing at the temperature and voltage that runs the lowest performance). Processors successfully tested for compliance with a given set of standards may be labeled with a higher clock rate, e.g., 3.50 GHz, while those that fail the standards of the higher clock rate yet pass the standards of a lesser clock rate may be labeled with the lesser clock rate, e.g., 3.3 GHz, and sold at a lower price.

The clock rate of a CPU is most useful for providing comparisons between CPUs in the same family. The clock rate is only one of several factors that can influence performance when comparing processors in different families. For example, an IBM PC with an [Intel 80486](https://en.wikipedia.org/wiki/Intel_80486" \o "Intel 80486) [CPU](https://en.wikipedia.org/wiki/Central_processing_unit) running at 50 MHz will be about twice as fast (internally only) as one with the same CPU and memory running at 25 MHz, while the same will not be true for MIPS R4000 running at the same clock rate as the two are different processors that implement different architectures and microarchitectures. Further, a "cumulative clock rate" measure is sometimes assumed by taking the total cores and multiplying by the total clock rate (e.g. dual core 2.8 GHz being considered processor cumulative 5.6 GHz).

The CPU time for a program is given by: CPU time = CPU clock cyles for a program × Clock cycle time.

Alternatively the CPU time can be measured as: CPU time = CPU clock cycles for a program / Clock rate .

CPU time depends on the program which is executed, including:

* the number of instructions executed, Computers are constructed is such way that events in hardware are synchronized using a clock
* types of instructions executed and their frequency of usage.

In order to measure the number of clock cycles I will use the Time-Stamp Counter Register. The Time-Stamp Counter (TSC) is a 64-bit model specific register (MSR) that accurately counts the cycles that occur on the processor. It is present on all x86 processors. The TSC is incremented every clock cycle and is set to zero every time the processor is reset. This register is accessible to the programmer since the Pentium processor.

To access the TSC, the programmer has to call the RDTSC (read time-stamp counter) instruction from assembly language. The RDTSC instruction loads the EDX:EAX with the content of the TSC register. The EDX will contain the high-order 32 bits and the EAX will contain the low-order 32 bits. The TSC counts the CPU cycles, so the value returned by the RDTSC instruction will be the number of cycles counted from the last processor reset to the point RDTSC was called.

This method for performance monitoring is very useful for measuring the cycle count for small sections of code. For example when trying to compare the performance of sections of code that have the same result but use different instructions. Another case in which this method can be of use is to obtain the average execution time for a function or section of code.

To obtain accurate results when measuring the performance with RDTSC instruction, the programmer has to be aware of the main issues that affect the cycle count and how to work-around these issues. The main issues that affect cycle count are:

• Out-of-order execution: the order of instruction execution is not as in the source code, this may cause the RDTSC instruction to return a cycle count that is less or greater than the actual cycle count for the measured sequence of code.

• Data cache and instruction cache: if the code or data are not in the cache, the cycle count is much larger.

• Context switches: if they occur during measurement, the result will be biased.

• Frequency changes: results are not accurate if there are frequency changes during measurement.

• Multi-core processors: the cycle counters on the cores are not synchronized. If the process migrates during measurement, the result will be wrong. The solutions for these issues are discussed as follows.

**3.2. FLOPS**

[Floating-point arithmetic](https://en.wikipedia.org/wiki/Floating-point_arithmetic) is needed for very large or very small [real numbers](https://en.wikipedia.org/wiki/Real_number" \o "Real number), or computations that require a large dynamic range. Floating-point representation is similar to scientific notation, except everything is carried out in base two, rather than base ten. The encoding scheme stores the sign, the [exponent](https://en.wikipedia.org/wiki/Exponent" \o "Exponent) (in base two for Cray and [VAX](https://en.wikipedia.org/wiki/VAX" \o "VAX), base two or ten for [IEEE floating point](https://en.wikipedia.org/wiki/IEEE_floating_point" \o "IEEE floating point) formats, and base 16 for [IBM Floating Point Architecture](https://en.wikipedia.org/wiki/IBM_Floating_Point_Architecture" \o "IBM Floating Point Architecture)) and the [Significand](https://en.wikipedia.org/wiki/Significand" \o "Significand) (number after the radix point). While several similar formats are in use, the most common is [ANSI/IEEE Std. 754-1985](https://en.wikipedia.org/wiki/IEEE_754-1985" \o "IEEE 754-1985). This standard defines the format for 32-bit numbers called single precision, as well as 64-bit numbers called double precision and longer numbers called extended precision (used for intermediate results). Floating-point representations can support a much wider range of values than fixed-point, with the ability to represent very small numbers and very large numbers.

The exponentiation inherent in floating-point computation assures a much larger dynamic range – the largest and smallest numbers that can be represented – which is especially important when processing data sets where some of the data may have extremely large range of numerical values or where the range may be unpredictable. As such, floating-point processors are ideally suited for computationally intensive applications.

FLOPS and [MIPS](https://en.wikipedia.org/wiki/Million_instructions_per_second" \l "Million_instructions_per_second" \o "Million instructions per second) are units of measure for the numerical computing performance of a computer. Floating-point operations are typically used in fields such as scientific computational research. The unit MIPS measures integer performance of a computer. Examples of integer operation include data movement (A to B) or value testing (If A = B, then C). MIPS as a performance benchmark is adequate when a computer is used in database queries, word processing, spreadsheets, or to run multiple virtual operating systems.[[3]](https://en.wikipedia.org/wiki/FLOPS" \l "cite_note-3)[[4]](https://en.wikipedia.org/wiki/FLOPS#cite_note-4) Frank H. McMahon, of the Lawrence Livermore National Laboratory, invented the terms FLOPS and MFLOPS (megaFLOPS) so that he could compare the supercomputers of the day by the number of floating-point calculations they performed per second. This was much better than using the prevalent MIPS to compare computers as this statistic usually had little bearing on the arithmetic capability of the machine.

**3.3. Memory Bandwidth**

Memory bandwidth is basically the speed of the video RAM. It's measured in gigabytes per second (GB/s). The more memory bandwidth you have, the better. A video card with higher memory bandwidth can draw faster and draw higher quality images. But there's more to video cards than just memory bandwidth. You also have to consider the drawing speed of the [GPU](http://www.playtool.com/pages/defs/defs.html#gpu). There's little point in getting a video card with a very fast GPU and limited memory bandwidth because the memory will be the bottleneck. The GPU will spend a lot of time doing nothing while waiting for its slow video RAM. By the same token, you don't want to get a video card with a slow GPU and very high memory bandwidth.

The memory bandwidth is determined by the memory clock, the memory type, and the memory width. The memory clock is the clock rate of the memory chips. Current (2006) memory chips have clock rates which range from about 167 [MHz](http://www.playtool.com/pages/defs/defs.html" \l "mhz" \o "MHz is short for megahertz - a megahertz is a million (1,000,000) cycles per second) to 1000 MHz. The most common memory type is double data rate [(DDR)](http://www.playtool.com/pages/defs/defs.html" \l "ddr" \o "double data rate) which means that it transfers two memory values for each memory clock cycle. There are also other kinds of DDR like DDR2, GDDR3, and GDDR4 and they also transfer at twice the memory clock rate. Some very old video cards still use single data rate (SDR) which transfers one value per clock cycle. The memory width of the common cards range from 32 bits to 256 bits. The maximum theoretical memory bandwidth is the product of the memory clock, the transfers per clock based on the memory type, and the memory width. For example, a video card with 200 MHz DDR video RAM which is 128 bits wide has a bandwidth of 200 MHz times 2 times 128 bits which works out to 6.4 [GB/s](http://www.playtool.com/pages/defs/defs.html" \l "gbs" \o "gigabytes per second). [This table](http://www.playtool.com/pages/vidtable/table.html) contains the video RAM bandwidth for many video cards in its [RAM speed](http://www.playtool.com/pages/vidtable/table.html" \l "tablerambandwidth) column. If you take a look at those memory bandwidths, you can see how much they vary between fast video cards and slow ones.

There are three different conventions for defining the quantity of data transferred in the numerator of "bytes/second":

1. The **bcopy convention**: counts the amount of data copied from one location in memory to another location per unit time. For example, copying 1 million bytes from one location in memory to another location in memory in one second would be counted as 1 million bytes per second. The bcopy convention is self-consistent, but is not easily extended to cover cases with more complex access patterns, for example three reads and one write.
2. The **STREAM convention**: sums the amount of data that the application code explicitly reads plus the amount of data that the application code explicitly writes.[[1]](https://en.wikipedia.org/wiki/Memory_bandwidth#cite_note-1) Using the previous 1 million byte copy example, the STREAM bandwidth would be counted as 1 million bytes read plus 1 million bytes written in one second, for a total of 2 million bytes per second. The STREAM convention is most directly tied to the user code, but may not count all the data traffic that the hardware is actually required to perform.
3. The **hardware convention**: counts the actual amount of data read or written by the hardware, whether the data motion was explicitly requested by the user code or not. Using the same 1 million byte copy example, the *hardware* bandwidth on computer systems with a [write allocate cache policy](https://en.wikipedia.org/wiki/CPU_cache) would include an additional 1 million bytes of traffic because the hardware reads the target array from memory into cache before performing the stores. This gives a total of 3 million bytes per second actually transferred by the hardware. The hardware convention is most directly tied to the hardware, but may not represent the minimum amount of data traffic required to implement the user's code.

I will use the third convention since it gives us the most information regarding the hardware. The code will consist of copying a large number of bytes from one memory location to another and measuring the time it took to perform this operation.

**3.4. Storage Device Speed**

Read & Write speeds are simply a measure of performance on a storage device. You are able to test the read/write speeds for all sorts of storage devices, whether they’re internal or external hard drives, solid-state drives, USB, SDs, CF’s etc.

Read & Write speeds are recorded with the letters “p/s (per second)” at the end of the measurement. For example, a device that has a write speed of 32 MBps means that it can record 32 MB (megabytes) of data every second.

When it comes to hard drive speed measurements, there are four that are important:

1. Sequential read speed, reading from a large block of contiguous data.
2. Sequential write speed, the same, but for writing data.
3. Random read speed, reading data scattered all over the disk.
4. Random write speed. Random speeds are generally far lower than sequential speeds because of the amount of seeking and rotational latency involved.

In order to perform the measurment I will design a program thar reads/ writes some data into/ from a file for a large number of times, then compute the time it took to do so.

1. **Implementation**
   1. **Clock frequency**

In order to measure the clock frequency I used a simple algorithm that counts the number of clock cycles it took to perform some function. That number is later divided by the number of clocks per second the system performs. The result is the time, in seconds, that it took to perform the operations.

The important piece of code is the following:

t = clock();

// some operations

t = clock() - t;

**4.2 FLOPS**

I used a similar method to measure the number of flops performed. The algorithm performs a number of operations on two random floating point variables and counts the time needed to perform each one, than it sums up the results.

int total = 0;

for(int i=0; i<n; i++){

float a, b;

int t = clock();

a \*= b;

total += clock() - t;

}

**4.3** **Bandwidth**

To measure the bandwidth I transferred a number of bytes from one place in memory to another, more specifically from one file to another, and computed the time needed to perform this operations.

t = clock();

for(int i=0; i<n; i++){

fin>>c;

fout<<c;

}

t = clock() - t;

The number of bytes that are transferred must be 3\*n because 3 operations are performed:

* Reading n bytes from fle
* Storing n bytes in a cache
* Writing n bytes in another file

**4.4** **Storage Device Speed**

I calculated the read and write speed of the storage device by reading a number of bytes from a file and writing them into another file and computing the time it took each operation separately.

for(int i=0; i<n; i++){

auxt1 = clock();

fin>>c;

t1 += clock() - auxt1;

auxt2 = clock();

fout<<c;

t2 += clock() - auxt2;

}

1. **Results**

These are the results of comparisons between the measurements done using c++ and those done using assembly.

**5.1 All meassurements in c++**

One conclusion that can be drawn is that floating point operations take a significant larger execution time than other types of operations.

**5.2 All measurements in assembly**

The results are similar with those in the previous graphic. In order to make a comparison, let’s see all of them together.

**5.3 All measurments**

As the graphic shows, the results of each pair of measurments are similar. In the case of flops the difference is notably bigger, which may be because it takes c++ more time to perform such operations.

Let’s have a better look at each of these pairs.

**5.4 Clock frequency**

Here you can see that the results obtained using the \_\_rdtsc() function in assembly are lower that those obtained using the clock() function in c++ with some exceptions.

**5.5 Flops**

Again, the measurements done in assembly show lower results, but they are closer to the average, which hints at the measurement in assembly being more precise.

**5.6 Bandwidth**

The measurements of bandwith show similar results both in c++ and assembly.

**5.7 Storage device speed**

Once again, the assembly measurement gives lower results than the c++ one, but this time the c++ measurement seems more precise.

1. **Conclusions**

More conclusions can be drawn from the results presented above. One of them is that the \_\_rdtsc() function is a bit faster than clock() function, hence the lower results obtained when measuring with the first one. The higher values appared when measuring floating point operations, suggesting that this type of operations take a significantly larger tyme to compute than other types of operations.

Also, the results obtained with \_\_rdtsc() are closer to the average, sugesting this is a more precise way of computing time than the c++ clock().

During the development of this project I learned the basics of measuring performance and the different benchmarks that can show the facilities of a hardware system.

I managed to develop few small benchmark agorithms that are capable of showing the performance of a computer and I saw the differece between different ways of measuring it.

1. **Bibliography**

* <https://en.wikipedia.org/wiki/Multi-core_processor>
* <https://www.nap.edu/read/12980/chapter/5>
* <https://en.wikipedia.org/wiki/Computer_performance>
* <https://www.tomshardware.com/news/clock-speed-definition,37657.html>
* <https://en.wikipedia.org/wiki/FLOPS>
* <https://www.lifewire.com/what-are-read-and-write-speeds-2640236>
* <https://en.wikipedia.org/wiki/Clock_rate>
* <http://u.cs.biu.ac.il/~wiseman/co/old/co1.pdf>
* <http://www.cse.ust.hk/~cktang/cs180/notes/lec03.pdf>
* <https://en.wikipedia.org/wiki/Memory_bandwidth>